МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В.Н. Каразіна

Факультет: **ННІ Каразінський банківський інститут**
Кафедра: **Інформаційних технологій та математичного моделювання**
Спеціальність: **122 Комп'ютерні науки**
Освітня програма: **Комп'ютерні науки та інформаційні технології в бізнесі**

Група: АК-21б **денна форма навчання**

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«СТВОРЕННЯ 2D ПЛАТФОРМЕРА З ВІДКРИТИМ СВІТОМ (MYSTERY OF MANSION)»
ЗА НАКАЗОМ № 4601-5/3045 ВІД 25 ВЕРЕСНЯ 2024 РОКУ

здобувача вищої освіти **Кулика Романа Володимировича**

Робота допущена до захисту в ЕК
протокол кафедри ІТММ № 4 від 30.11.2024р.

Завідувач кафедри ІТММ
к.п.н., доцент
_____ Н. І. Стяглик

Науковий керівник
к.ф.-м.н., доцент
_____ Л. Д. Філатова

м. Харків 2024 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В. Н. Каразіна

Факультет навчально-науковий інститут "Каразінський банківський інститут"
Кафедра інформаційних технологій та математичного моделювання
Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри

Н. І. Стяглик
"25" вересня 2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ (ПРОЄКТ)

Кулик Роман Володимирович
(прізвище, ім'я, по батькові студента)

1. Тема роботи Створення 2D платформера з відкритим світом (MYSTERY of MANSION)
керівник роботи Філатова Любов Дмитрівна к.ф.-м.н.,доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "25" вересня 2024 року № 4601-5/3045

2. Строк подання студентом роботи 20 листопада 2024 року

3. Перелік питань, які потрібно розробити:

У розділі 1: Аналіз вимог до програмного продукту

- Які ключові механіки повинні бути реалізовані в 2D-платформері?
- Як Python та його бібліотеки (наприклад, Pygame) сприяють створенню платформерів?
- Які переваги має бібліотека Tiled для створення рівнів?

Урозділі 2: Проєктування програмного продукту

- Які ключові технічні вимоги слід враховувати при розробці "MYSTERY of MANSION"?
- Як діаграми UML можуть допомогти у відображенні архітектури гри?
- Як забезпечити багатоплатформність гри?

Урозділі 3: Конструювання програмного забезпечення

- Які функції бібліотеки Pygame найкраще підходять для розробки ігрових механік?
- Як створити ефективний код для інтеграції графіки та фізичних об'єктів?
- Які методи використовувати для забезпечення безпеки та стабільності гри?

Урозділі 4: Економічне обґрунтування

- Які основні витрати передбачаються для розробки гри?
- Які моделі монетизації доцільно застосувати для проекту?
- Як оцінити економічну ефективність проекту порівняно з подібними іграми?

4. План роботи

№ з/п	Назви етапів роботи
1	Вибір здобувачем теми кваліфікаційної магістерської роботи
2	Затвердження плану і завдання кваліфікаційної магістерської роботи
3	Здача кваліфікаційної магістерської роботи керівнику
4	Підпис кваліфікаційної магістерської роботи керівника
5	Підпис кваліфікаційної магістерської роботи у нормоконтролера
6	Допуск завідувачем кафедри до захисту кваліфікаційної магістерської роботи
7	Захист кваліфікаційної магістерської роботи

5. Дата видачі завдання 25 вересня 2024 року

Студент _____ Кулик Р.В
підпис ініціали, прізвище

Керівник роботи _____ Філатова Л.Д
підпис ініціали, прізвище

РЕФЕРАТ
НА КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ
«СТВОРЕННЯ 2D ПЛАТФОРМЕРА З ВІДКРИТИМ СВІТОМ (MYSTERY of MANSION)»
Кулика Романа Володимировича

Кваліфікаційна магістерська робота містить 66 сторінок, 12 таблиць, 18 рисунків, список літератури з 40 найменувань.

Об′єктом дослідження є процес створення двовимірної гри-платформера, реалізованої з використанням мови програмування Python, який включає як технічні аспекти розробки, так і реалізацію інтерактивних механік, характерних для жанру платформерів.

Предметом дослідження є специфіка використання програмного модуля pygame для розробки ігрового функціоналу, можливості середовища розробки Python, а також графічного редактора Tiled, що застосовується для створення та налаштування візуального оформлення ігрових рівнів.

Мета кваліфікаційної магістерської роботи полягає у розробці повноцінного програмного продукту, що відображає концепцію класичного платформера, а також у визначенні шляхів для його вдосконалення та масштабування, враховуючи потреби користувачів і потенціал ринку..

Завданнями кваліфікаційної магістерської роботи є:

- детально дослідити та визначити поняття «платформер», а також проаналізувати тенденції розвитку ігор цього жанру та можливості їх інтеграції в сучасний ігровий ринок;
- обґрунтувати актуальність розробки даного проекту, враховуючи інтерес до 2D ігор та постійний попит на доступні й прості в освоєнні ігрові продукти;
- розробити технічні та функціональні вимоги до програмного продукту, базуючись на порівнянні з існуючими аналогами;
- створити основні ігрові механіки та розробити візуальні елементи продукту, що забезпечать привабливість та інтерактивність гри;
- провести попередній економічний розрахунок та оцінити потенційні комерційні вигоди від реалізації проекту.
- протестувати, проаналізувати та за потреби виправити недоліки коду які можуть з'явитися в ході розробки, перед його остаточним завершеням.

Актуальність дослідження: обумовлена постійним зростанням популярності комп'ютерних ігор, які мають значний вплив на сучасну культуру, розважальний сектор та економіку. Ігри, зокрема жанру платформер, привертають увагу гравців завдяки доступності, простоті ігрового процесу та можливості їхнього використання на різних платформах. Крім того, розвиток технологій, що підвищують якість ігрового досвіду, робить проектування таких продуктів важливим і перспективним напрямом.

За результатами дослідження: нами було сформульовано низку теоретичних і практичних рекомендацій для створення 2D гри-платформера,

які мають прикладний характер і можуть бути використані як орієнтир для розробників при впровадженні подібних проектів.

Практична новизна: проекту полягає в його мультиплатформеності, тобто можливості використовувати гру на різних пристроях і платформах, таких як мобільні телефони, персональні комп'ютери та ігрові консолі. Такий підхід забезпечує ширший доступ для різних категорій користувачів, що збільшує потенційну аудиторію гри та підвищує її комерційну привабливість.

Одержані результати можуть бути використані як основа для подальшого розвитку і вдосконалення створеного продукту, зокрема для його комерційного розповсюдження, додавання нових рівнів, покращення графіки, інтеграції додаткових функцій та адаптації для нових платформ.

КЛЮЧОВІ СЛОВА: ПРОГРАМНИЙ ПРОДУКТ, PYTHON, ПЛАТФОРМА, МУЛЬТИПЛАТФОРМЕНІСТЬ, КОМЕРЦІЙНИЙ ПОТЕНЦІАЛ, СИСТЕМА, РОЗВИТОК, МОДЕРНІЗАЦІЯ, ІНТЕРАКТИВНІСТЬ.

ABSTRACT
AT QUALIFICATION BACHELOR WORK
"CREATION OF A 2D PLATFORMER WITH AN OPEN WORLD (MYSTERY of MANSION)"
Roman Kulyk

The master's thesis contains 66 pages, 12 tables, 18 drawings, a list of references of 40 titles.

The object of the process of creating a 2D platformer game implemented using the Python programming language, which includes both the technical aspects of development and the implementation of interactive mechanics typical of the platformer genre.

The subject of the specifics of using the Pygame module for developing game functionality, the capabilities of the Python development environment, and the Tiled graphic editor, which is used for creating and configuring the visual design of game levels.

The purpose of involves developing a full-fledged software product that reflects the concept of a classic platformer, as well as identifying ways to improve and scale it, taking into account user needs and market potential.

The tasks of a bachelor's degree are:

- Thoroughly research and define the concept of a "platformer," as well as analyze the development trends of games in this genre and their potential integration into the modern gaming market.
- Justify the relevance of developing this project, considering the interest in 2D games and the ongoing demand for accessible and easy-to-learn gaming products.
- Develop technical and functional requirements for the software product, based on comparisons with existing counterparts.
- Create the core game mechanics and design visual elements that ensure the game's appeal and interactivity.
- Conduct a preliminary economic assessment and evaluate potential commercial benefits from implementing the project.
- Test, analyze, and, if necessary, correct code issues that may arise during development before final completion.

The relevance driven by the continuous growth in the popularity of video games, which have a significant impact on modern culture, the entertainment sector, and the economy. Games, particularly in the platformer genre, attract players due to their accessibility, simplicity of gameplay, and compatibility across different platforms. Moreover, advancements in technologies that enhance the gaming experience make the design of such products an important and promising field.

According to the results of the research: we have formulated a series of theoretical and practical recommendations for creating a 2D platformer game, which are applied in nature and can serve as a guide for developers in

implementing similar projects.

Main theoretical provisions on the topic of the practical relevance of the project's strength lies in its multiplatform capability, meaning the game can be used on various devices and platforms, such as mobile phones, personal computers, and gaming consoles. This approach provides broader access for different user categories, increasing the game's potential audience and enhancing its commercial appeal.

The results obtained can be used as a foundation for further development and improvement of the created product, particularly for its commercial distribution, adding new levels, enhancing graphics, integrating additional features, and adapting it for new platforms.

KEYWORDS: SOFTWARE PRODUCT, PYTHON, PLATFORM, MULTIPLATFORM CAPABILITY, COMMERCIAL POTENTIAL, SYSTEM, DEVELOPMENT, MODERNIZATION, INTERACTIVITY.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧОК, СИМВОЛІВ І ТЕРМІНІВ

- **PZ** – Програмне забезпечення
- **UML** – Уніфікована мова моделювання
- **IDLE** – Інтегроване середовище розробки Python
- **PC** – Персональний комп’ютер
- **UI** – Інтерфейс користувача (User Interface)
- **QA** – Забезпечення якості (Quality Assurance)
- **TBD** – Технічне завдання
- **SKUD** – Система контролю та управління доступом
- **WBS** – Структура робіт (Work Breakdown Structure)

ВСТУП

Індустрія комп'ютерних ігор з'явилася порівняно нещодавно, але за короткий проміжок часу вона перетворилася на галузь з великими фінансовими доходами. Розповсюдження комп'ютерних технологій, зокрема Інтернету, стало основним фактором, що сприяло стрімкому зростанню популярності віртуальних розваг. Ігри стали набагато доступнішими для користувачів завдяки можливості запускати їх на комп'ютерах та ігрових приставках, а також через цифрові платформи для дистрибуції. Це дало змогу значно розширити аудиторію, адже тепер для того, щоб грати в комп'ютерні ігри, достатньо мати відповідне обладнання та доступ до Інтернету, а також не потрібно мати спеціальних знань для вибору ігор[10].

Сьогодні комп'ютерні ігри вже не обмежуються лише розвагою. Вони активно використовуються в освітніх і навчальних цілях, наприклад, для створення симуляторів, які допомагають навчанню фахівців у різних сферах, таких як пілоти, хірурги, військові тощо [31]. Це відкриває нові можливості для вдосконалення професійних навичок через ігрові технології.

Щодо нашої країни, то хоча ігрова індустрія тут розвивається повільніше порівняно з іншими державами, її потенціал є значним. Однією з причин цього є пізніше впровадження комп'ютерних розваг у культуру країни та обмежена кількість компаній-розробників. Однак з розвитком технологій та зростаючим інтересом до комп'ютерних ігор, цей сектор має великі перспективи. У порівнянні з міжнародним ринком, українська індустрія комп'ютерних ігор була значно відставала, але з часом з'являються нові компанії, готові конкурувати на цьому ринку. Попит на ігри зростає, але розробники поки що не мають великої кількості конкурентоспроможних продуктів, що відкриває широкі можливості для подальшого розвитку.[12]

Важливим напрямом розвитку є створення програмного забезпечення для ігор, особливо для 2D-платформерів. Ці ігри зазвичай є простими у виконанні і зрозумілими для гравця, що робить їх дуже привабливими для

широкої аудиторії. Розробка таких ігор не потребує великих затрат часу на навчання, а геймплей легко освоюється [33].

Головним об'єктом дослідження є процес створення комп'ютерних ігор, а предметом - технології, що використовуються для розробки 2D-платформерів. В умовах швидкого розвитку ігрової індустрії та зростаючого числа активних гравців, ринок відеоігор продовжує приносити величезні прибутки. За даними, на кінець 2023 року, прибуток від індустрії ігор перевищив 150 мільярдів доларів, що робить цей сектор одним з найперспективніших напрямків у галузі програмного забезпечення[11].

Сучасне суспільство перенасичене великими проектами, які вимагають значних часових затрат. В умовах постійної зайнятості багатьох людей, їм важко знайти достатньо часу для глибокого занурення в складні ігрові проекти. Тому ігри з простим і доступним геймплеєм, такі як 2D-платформери, стали дуже популярними серед людей, які мають обмежений час для розваг. Вони дозволяють швидко насолоджуватися ігровим процесом без потреби витрачати великі кількості часу [32].

РОЗДІЛ 1. АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ПРОДУКТУ

1.1. Формування мети та аналіз предметної області

Метою цієї дипломної роботи є створення прототипу комп'ютерної гри в жанрі 2D-платформер, що реалізується за допомогою мови програмування Python. Жанр платформерів є одним з найбільш популярних та улюблених серед геймерів завдяки своїй простоті, доступності та захоплюючому геймплею. Платформери характеризуються тим, що вони не вимагають від гравців глибоких технічних знань, тому вони підходять для широкої аудиторії, включаючи людей з різним рівнем досвіду в ігровій індустрії. Вони ідеально підходять для тих, хто шукає легке, але захоплююче проведення часу, з можливістю взаємодії з інтуїтивно зрозумілими механіками[9].

Платформери мають низький поріг входження, що дозволяє навіть новачкам швидко освоїтися в грі, а в той же час створюють можливості для розвитку та вдосконалення навичок з часом. Оскільки ці ігри, зазвичай, не мають складних сюжетних ліній чи складної механіки, вони можуть бути відмінним варіантом для тих, хто хоче просто насолодитися простим, але захоплюючим ігровим процесом. Цей жанр має великий потенціал для реалізації різноманітних ідей та концепцій, що робить його дуже гнучким і дозволяє створювати як легкі розважальні ігри, так і більш складні проекти з цікавими механіками і глибоким змістом [34].

Розробка прототипу 2D-платформера за допомогою мови Python є захоплюючим і технологічно цікавим завданням. Python — це високорівнева мова програмування, яка отримала визнання завдяки своїй простоті та багатофункціональності. Однією з головних переваг Python є велика кількість бібліотек, фреймворків та інструментів, що можуть бути використані для створення різноманітних ігор. Однією з найбільш відомих та популярних бібліотек для створення ігор є Pygame. Вона дозволяє ефективно працювати з

графікою, аудіо, а також організовувати взаємодію з користувачем через введення з клавіатури або миші, що робить її ідеальним інструментом для створення 2D-ігор[1].

Під час розробки гри буде особлива увага приділятися створенню захоплюючих рівнів з платформами різної висоти, складності та різноманітними перешкодами. Кожен рівень буде вимагати від гравця вміння точно планувати свої стрибки, правильно розраховувати відстані та шукати найефективніші способи подолання бар'єрів. Для цього будуть створюватися випробування, які випробують здатність гравця адаптуватися до змінюваних умов та використовувати різні механіки гри, зокрема стрибки, рух по платформам, взаємодія з об'єктами та багато іншого.

Однією з основних особливостей гри стане наявність різноманітних перешкод, таких як шипи, рухомі платформи, лазери, пастки та інші небезпеки. Гравець буде змушений уважно спостерігати за ситуацією на кожному етапі та приймати швидкі рішення, чи уникати перешкоди, чи розробляти стратегію для їх подолання. Це додасть грі елемент головоломки, адже гравець постійно буде стикатися з різними видами перешкод і потребуватиме постійної концентрації та аналітичного підходу[2].

Ще однією важливою механікою гри стане система збирання предметів, яка мотивуватиме гравця досліджувати кожен закуток гри і шукати бонуси або важливі ресурси, без яких в деякі місця просто не дістатися. Гравець зможе збирати монети або ключі, що дасть йому можливість просуватись до фіналу, отримати різноманітні бонуси або отримати доступ до нових можливостей гри [35]. Це буде не лише стимулювати до виконання основних завдань гри, але й заохочувати до дослідження кожної локації на предмет прихованих секретів[4].

Що стосується ворогів, вони будуть мати різні характеристики та поведінкові шаблони. Деякі вороги зможуть атакувати гравця, випускаючи шипи або стріляючи снарядами. Інші можуть бути більш пасивними, але небезпечними через свої рухи або схованки. Кожен ворог вимагатиме

окремого підходу і тактики для подолання, що зробить бої більш захоплюючими та різноманітними. Для боротьби з ними гравець зможе використовувати свої навички стрибків, а також різноманітні інструменти або зброю, якщо вони будуть доступні [40].

Графічно гра буде виконана в мальованому мультяшному стилі, що надасть їй унікальний вигляд. Стиль графіки буде яскравим та барвистим, а персонажі будуть виконані у вигляді міфічних істот або антропоморфних тварин, що дозволить додати грі індивідуальності та відповідати характеру жанру. Це забезпечить яскравий та незабутній візуальний досвід для гравців.

Успішне завершення цієї дипломної роботи дозволить не тільки продемонструвати набуті знання з розробки ігор, а й створити прототип, який може стати основою для подальшої роботи над повноцінною грою. Такий проект має значний потенціал для подальшого розвитку, як у плані розширення контенту, так і для можливості комерційного використання [38].

1.2. Інструменти для розробки 2D платформерів на Python

ІТ-інструменти займають важливе місце у роботі розробників ігор, оскільки вони значно спрощують створення складних механік, графіки, анімацій та звукових ефектів. Розглянемо кілька основних інструментів, що використовуються для розробки 2D платформерів на Python [3].

Pygame є однією з найбільш поширених бібліотек для створення 2D ігор на Python. Вона дозволяє ефективно працювати з графікою, звуком та обробкою подій, таких як натискання клавіш або рух миші. Бібліотека базується на SDL (Simple DirectMedia Layer), що забезпечує доступ до важливих мультимедійних функцій, таких як рендеринг графіки та відтворення звуків [6].

Основні переваги Pygame включають:

- Простота використання: бібліотека має інтуїтивно зрозумілий інтерфейс і легку документацію, що робить її доступною для новачків.

- Гнучкість: Pygame надає широкі можливості для реалізації різноманітних ігрових механік і елементів, що дозволяє створювати унікальні концепції.
- Кросплатформність: бібліотека підтримує роботу на різних операційних системах, таких як Windows, macOS і Linux, що робить її універсальним інструментом для розробки.

При використанні Pygame розробник може застосовувати готові модулі для відображення спрайтів, взаємодії з користувачем, обробки зіткнень і анімації. Це значно скорочує час на реалізацію базових механік, що робить процес розробки гри швидшим і ефективнішим.

Arcade є ще однією популярною бібліотекою для створення 2D ігор на Python, яка пропонує більш сучасний підхід порівняно з Pygame. Вона використовує OpenGL для рендерингу графіки, що дозволяє створювати візуально привабливі та продуктивні ігри. Arcade підтримує безліч функцій для створення платформерів, таких як система анімації, рендеринг графіки, фізичні зіткнення та інші [5].

Переваги Arcade:

- Продуктивність: завдяки використанню OpenGL, Arcade має високу швидкість обробки графіки, що дозволяє створювати насичені ігри.
- Простота: бібліотека орієнтована на простоту використання, що робить її доступною для початківців і дозволяє швидко розпочати розробку ігор.
- Підтримка фізики: Arcade має вбудовану підтримку фізичних ефектів, таких як гравітація та зіткнення, що значно полегшує реалізацію складних ігрових механік.

Godot Engine є потужною платформою для розробки ігор, хоча вона не написана на Python. Однак Godot підтримує інтеграцію через GDScript, який базується на Python, що дозволяє Python-розробникам використовувати цей движок для створення високоякісних 2D та 3D ігор. Godot пропонує інтуїтивно зрозумілий інтерфейс, потужні інструменти для створення

анімацій та фізичних ефектів, а також підтримує кросплатформність [4].

Переваги Godot:

- Гнучкість: Godot дозволяє створювати різноманітні ігри завдяки потужному скриптовому движку та можливості візуального програмування.
- Безкоштовність: Godot є безкоштовним і відкритим програмним забезпеченням, що робить його доступним для різних розробників.
- Командна робота: Godot підходить для командних проектів, де необхідна інтеграція різних аспектів, таких як графіка, звукове оформлення та мережеві технології.

Розробка 2D платформера включає кілька ключових етапів, кожен з яких вимагає застосування різних IT-інструментів. Ось основні етапи розробки:

- Концептуалізація гри: на цьому етапі розробник визначає основну ідею гри, враховуючи цільову аудиторію, використовувані механіки і бажаний ігровий досвід. Розробка документації, що включає концепт-арт, опис механік і персонажів, дозволяє визначити загальний план і розвиток проекту.
- Створення прототипу: після визначення концепції розробник створює базовий прототип гри, в якому тестуються основні механіки, такі як стрибки, рухи персонажа і взаємодія з об'єктами. Використання інструментів для швидкого створення прототипів, таких як Pygame або Arcade, дозволяє отримати зворотний зв'язок щодо основних елементів гри і виявити проблеми на ранніх етапах розробки.
- Створення рівнів: створення рівнів є важливим етапом розробки 2D платформера. Гра повинна включати різноманітні рівні з поступовим збільшенням складності, що заохочують гравців досліджувати нові механіки. Для цього можна використовувати інструменти, такі як Tiled, для створення рівнів за допомогою тайлових карт, які інтегруються з кодом на Python через Pygame або Arcade.
- Графічне оформлення та анімація: графічне оформлення є

важливою частиною гри. Для створення піксельної графіки та анімацій персонажів використовуються інструменти, такі як Aseprite. Важливо забезпечити плавність анімацій, щоб не відволікати гравця від ігрового процесу і зробити персонажів більш живими та привабливими.

- Звукове оформлення: звукове оформлення додає грі емоційної глибини і забезпечує захоплюючий досвід. Інструменти, такі як Audacity або FMOD, дозволяють створювати і редагувати звукові ефекти для інтеграції у гру через Pygame. Важливо враховувати синхронізацію звуку з ігровими подіями, що підвищує емоційну насиченість гри.
- Тестування: після завершення розробки гри проводиться тестування всіх аспектів проекту, включаючи функціональне тестування механік, тестування на продуктивність і виявлення помилок. Для автоматизованого тестування можна використовувати бібліотеки, такі як unittest. Бета-тестування з реальними користувачами дозволяє отримати цінний зворотний зв'язок і виявити можливі недоліки перед випуском гри.

1.3. Огляд існуючих аналогів

Платформери, попри збереження основних елементів, продовжують розвиватися, впроваджуючи нові механіки та ідеї. Завдяки цьому в жанрі створюються різноманітні й захоплюючі ігри, які мають свою аудиторію серед гравців. Важливу роль у розвитку цього жанру відіграють такі ігри, що стали знаковими й допомогли розширити горизонти платформерів. Ось декілька ключових представників цього жанру [36].

Jumping Frog (1978)

"Jumping Frog" (рис. 1.1) вважається однією з перших ігор, де головний персонаж мав здатність до стрибків, хоч і без традиційних платформ [13]. Ця нова механіка зробила гру новаторською для свого часу та вплинула на подальший розвиток жанру, надихнувши розробників на створення більш динамічних ігрових механік.

Space Rescue (1980)

У "Space Rescue"(рис. 1.2) було додано платформу для пересування, що стало важливим нововведенням. Гра ще не мала функції стрибка, яка стала типовою для жанру платформерів згодом [14]. Однак сам факт використання

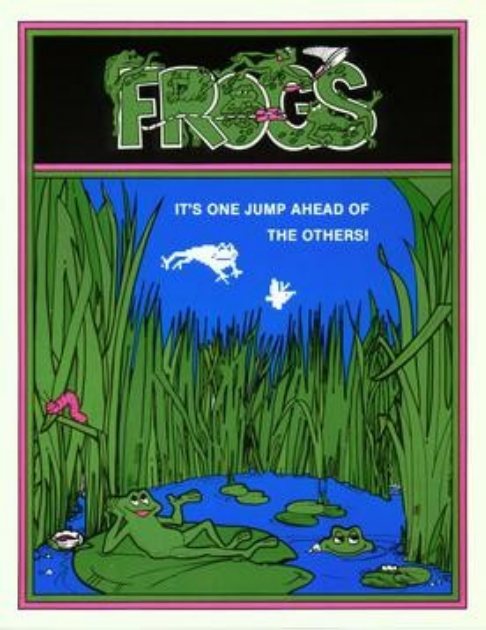


Рис. 1.1. Гра Jumping Frog

платформи додав новий вимір до геймплею, прокладаючи шлях до складніших ігор.

Рис. 1.2. Гра Space Rescue

King Kong's Revenge (1981)

"King Kong's Revenge" (рис. 1.3) представила концепцію скролінгу

екрану, дозволяючи гравцю рухатися вгору або вниз [13]. Це додало нову динаміку до гри та стало важливим елементом у розвитку платформерів, значно розширюючи варіативність геймплею.

Рис. 1.3. Гра King Kong's Revenge

Super Adventure Quest (1985).

"Super Adventure Quest" (рис. 1.4) встановила нові стандарти у жанрі платформерів. Вперше гравець отримав можливість не лише стрибати, але й збирати різні предмети та боротися з ворогами [14]. Це значно збагатило геймплей і стало еталоном для майбутніх розробок у жанрі.

Рис. 1.4. Гра Super Adventure Quest

Перші платформери мали обмежені, статичні ігрові світи, зазвичай із профільною перспективою. Персонажі могли стрибати між платформами, збирати бонуси та долати ворогів, але ігри були доволі простими [15]. Згодом

із розвитком технологій платформери стали більш динамічними та отримали горизонтальне прокручування, що відкрило нові можливості.

Pitfall! (1982).

Один з піонерів жанру, "Pitfall!" (рис. 1.5) запропонував гравцям відкритий, рухомий світ. Завдяки своїм можливостям для дослідження, проходження перешкод та збирання скарбів гра стала новаторською, надихнувши цілу низку подальших ігор [14].

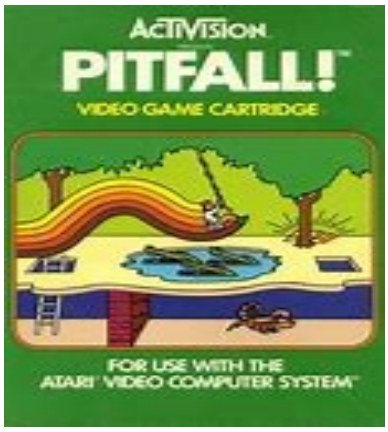


Рис. 1.5. Гра Pitfall!

Manic Miner (1983) і Jet Set Willy (1984) (рис. 1.6).

Ці класичні платформери здобули популярність на домашніх комп'ютерах, пропонуючи гравцям унікальні дизайни рівнів та головоломки, які вимагали обдуманого підходу і зробили ці ігри справжніми хітами свого часу.

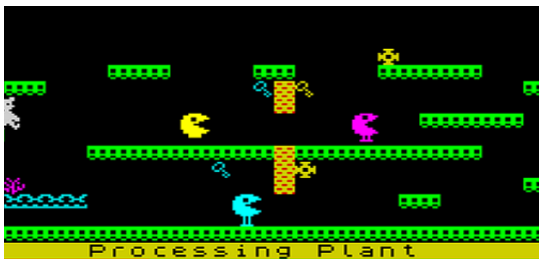


Рис. 1.6. Гра Manic Miner і Jet Set Willy

Super Mario Bros. (1985).

Цей платформер від Nintendo став знаковою грою завдяки своїм

масштабним і захопливим рівням. "Super Mario Bros." (рис. 1.7) закріпила стандарти для жанру, а персонаж Маріо став впізнаваним символом не лише для Nintendo, але й для всієї ігрової індустрії [13].

Рис. 1.7. Гра Super Mario Bros

Castlevania (1987) (рис. 1.8).

У цій грі жанр платформерів збагатився фентезійним стилем, надавши гравцям можливість боротися з вампірами та іншими монстрами у готичному середовищі замку, що стало своєрідною візитівкою серії [14].

Рис. 1.8. Гра Castlevania

Sonic the Hedgehog (1991).

"Sonic the Hedgehog" (рис. 1.9) від Sega додала до жанру швидкість, яскравий динамізм та унікальні механіки, такі як гоночні сегменти, збирання кілець і спеціальні здібності Соніка [15]. Ця гра стала однією з найуспішніших в індустрії.

Рис. 1.9. Гра Sonic the Hedgehog

Платформери 3D-епохи.

З переходом до 3D платформери стали масштабнішими і різноманітнішими. Ігри на кшталт "Crash Bandicoot" і "Spyro the Dragon" впровадили нові механіки та нелінійні сюжети, де гравці могли вільно досліджувати великі світи [13]. Ці нововведення надали жанру нового життя і зробили його ще більш захопливим. Приклад наведено на рисунку 1.10.

Рис. 1.9. Гра Crash Bandicoot і Spyro the Dragon

РОЗДІЛ 2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

2.1. Оформлення технічного завдання

Технічне завдання для створення гри-платформера "MYSTERY of MANSION" охоплює широке коло вимог, які забезпечать успішну розробку і повноцінне функціонування продукту.Основні вимоги розберемо нижче:

1. Призначення гри.

MYSTERY of MANSION — це захоплююча розважальна гра, у якій гравець досліджує таємничий особняк, зустрічає різноманітних монстрів і долає рівні, багаті на загадки та випробування [37]. Основна мета гравця полягає у виживанні та подоланні всіх викликів, що трапляються на його шляху. Гра спрямована на розвиток навичок стратегічного мислення та швидкої реакції, оскільки гравцеві потрібно знайти правильні рішення для кожного рівня, використовуючи доступні ресурси та ігрові предмети. Гра обіцяє створити для гравця атмосферу інтриги та загадковості, які характерні для історії та оточення старого особняка [8].

2. Цільова аудиторія.

MYSTERY of MANSION призначена для гри одним користувачем, орієнтованим на цікаве проведення часу. Гра має бути зручною для гравців різного віку та ігрового досвіду, включаючи як новачків, так і більш досвідчених користувачів. Для цього інтерфейс і ігрові елементи мають бути інтуїтивно зрозумілими та не перевантаженими зайвою інформацією.

3. Параметри налаштування.

Щоб надати гравцеві можливість адаптувати гру під власні потреби та вподобання, передбачені різні налаштування звукового супроводу. До них належать:

- Регулювання гучності: гравець може змінювати рівень гучності звуків і музики, щоб створити оптимальний для себе ігровий досвід.
- Вимкнення звуку: для тих, хто віддає перевагу грі без звукових

ефектів, доступна можливість повного вимкнення звуку.

- Зміна розміру екрану для більшого занурення чи навпаки комбінування з іншими процесами.
- Вибір мови як текстової частини так і голосового супровіду.

Ці налаштування забезпечують комфортний процес гри, дозволяючи гравцям підлаштувати гру до їхніх уподобань або конкретних ситуацій.

4. Вимоги до проходження рівнів.

Усі рівні гри мають бути логічно структурованими та прохідними. Вони повинні пропонувати гравцеві виклики, які можна подолати за умов правильного використання доступних ігрових ресурсів. Це означає, що дизайн рівнів передбачає баланс між складністю та досяжністю цілей, дозволяючи гравцям поступово вдосконалювати свої навички та адаптуватися до нових викликів у міру проходження гри [9].

5. Функціональні можливості гри.

Основні функції, які повинна підтримувати MYSTERY of MANSION, включають:

- Запуск гри: гравець може швидко почати нову гру або продовжити попередню.
- Збереження та вихід: можливість безпечного збереження прогресу та виходу з гри.
- Перегляд результатів рівня: надає статистику проходження кожного рівня, включаючи кількість зібраних ігрових предметів та знищених монстрів.
- Боротьба з монстрами: реалізована механіка бою з різноманітними типами монстрів, яких гравець зустрічатиме на рівнях. Кожен монстр володіє унікальними характеристиками, що робить їхнє подолання цікавим і різноманітним.
- Збирання ігрових предметів (ітемів): у грі передбачено різні типи предметів, наприклад, "життя", які дозволяють відновити здоров'я, та "монети", що можуть бути використані для різних цілей або накопичуватися

для досягнення більш високих результатів.

- Увімкнення/вимкнення звукового супроводу: можливість контролювати музичний фон і звукові ефекти без необхідності виходу з гри.
- Вибір мови.

Гра не вимагає введення текстових даних, а вся навігація та керування реалізуються через ігрові контролери або клавіатуру, що забезпечує плавний ігровий процес.

6. Конфіденційність даних.

Гра не збирає та не зберігає конфіденційну інформацію користувачів, гарантуючи безпечність персональних даних. MYSTERY of MANSION розроблена з урахуванням високих стандартів конфіденційності та не взаємодіє з інформацією, що може ідентифікувати користувача [20].

7. Вимоги до обслуговування та підтримки.

MYSTERY of MANSION не потребує специфічного технічного обслуговування. Надійність функціонування програмного продукту забезпечується регулярним технічним обслуговуванням комп'ютерного обладнання, включаючи безперебійне електроживлення та періодичне сканування антивірусним програмним забезпеченням.

Вимоги до обладнання.

Гра призначена для встановлення на ПК з мінімальними системними вимогами:

- Операційна система: Windows 7 або новіша версія.
- Оперативна пам'ять: щонайменше 256 МБ, рекомендовано 512 МБ для забезпечення стабільності.
- Роздільна здатність екрану: 1024x768 або вища.
- Середовище Python (версія 2.11.4 або вище) або альтернативи — Jython, PyPy чи IronPython.

Вимоги до сумісності

Програмне забезпечення MYSTERY of MANSION має бути сумісним з операційною системою Windows 7 і вище. Воно не повинно вступати у

конфлікт з іншими програмами, встановленими на комп'ютері користувача, забезпечуючи коректну роботу в умовах різноманітного програмного середовища [39].

Документація для користувача.

Детальна документація включатиме:

- Інструкції з установки: покроковий опис процесу встановлення.
- Керівництво по запуску: інструкції для початку гри.
- Опис функцій: короткий огляд основних функцій та елементів керування.
- Налаштування: інформація про доступні налаштування та їхню конфігурацію.
- Вирішення проблем: список можливих проблем, з якими може зіткнутися гравець, та способи їх вирішення.

Тестування і технічна документація.

Технічне завдання слугує основним документом, що описує вимоги до гри MYSTERY of MANSION, включаючи всі аспекти її функціональності, інтерфейсу та методів тестування. У документі з тестування будуть зафіксовані результати тестів, проведені сценарії та виявлені помилки, а також способи їх усунення. Ці документи сприятимуть узгодженій роботі між розробниками та замовником і полегшать експлуатацію гри користувачем.

2.2. Використання UML для побудови діаграм

Уніфікована мова моделювання (UML) є всесвітньо прийнятим стандартом, який широко використовується в розробці програмного забезпечення для опису, проектування та документування програмних систем. Завдяки UML розробники та аналітики можуть створювати графічні моделі, які описують різні компоненти системи, їхні функції, взаємозв'язки та поведінку, що в результаті дозволяє оптимізувати процес розробки та

управління складними проєктами [10].

Основною метою UML є забезпечення уніфікації та стандартизації мови моделювання. Це дозволяє зменшити ризики непорозумінь та підвищити ефективність співпраці між розробниками, системними аналітиками, бізнес-аналітиками, архітекторами програмного забезпечення та іншими зацікавленими сторонами проєкту. Стандартизована мова UML надає набір чітких та зрозумілих термінів, символів і концепцій, які полегшують комунікацію в команді, сприяють ефективному обговоренню проєкту та дозволяють кожному учаснику розуміти загальну картину системи [11].

UML включає в себе набір діаграм, кожна з яких призначена для представлення певних аспектів системи. Ці діаграми дозволяють сфокусуватися на конкретних аспектах або деталях системи та полегшити їх подальшу реалізацію. Серед основних типів UML-діаграм можна виділити такі:

- Діаграми класів: використовуються для моделювання структури класів, атрибутів, методів та їх взаємозв'язків. Цей тип діаграм відображає статичну структуру системи та визначає основні класи, а також зв'язки між ними, такі як успадкування та асоціація.
- Діаграми послідовності: допомагають моделювати порядок взаємодії об'єктів у процесі виконання певного сценарію або послідовності подій. Це дозволяє зрозуміти логіку взаємодії між об'єктами та забезпечити цілісність процесів у рамках певного функціоналу системи.
- Діаграми діяльності: призначені для моделювання бізнес-процесів, процедур та алгоритмів, дозволяючи відображати потоки робіт і альтернативні сценарії виконання процесів. Діаграми діяльності корисні для аналізу робочих процесів та підвищення їх ефективності.
- Діаграми компонентів: використовуються для опису програмних компонентів, таких як модулі, бібліотеки, файли та інші частини системи, а також їх взаємозалежностей. Вони дозволяють наочно продемонструвати, як компоненти взаємодіють один з одним у рамках загальної архітектури.

- Діаграми розгортання: моделюють фізичне розміщення компонентів системи на апаратних засобах, таких як сервери або клієнтські машини. Це дозволяє оцінити архітектуру розподіленої системи, планувати ресурси та оптимізувати продуктивність.
- Діаграми прецедентів: показують взаємодію між акторами (користувачами, системами або зовнішніми сутностями) та функціональністю системи. Вони дають змогу визначити та проаналізувати вимоги до системи, описуючи основні сценарії взаємодії з нею.
- Діаграми станів: моделюють поведінку об'єктів залежно від їх поточного стану та подій, які можуть впливати на цей стан. Ці діаграми використовуються для відображення життєвого циклу об'єкта та дозволяють врахувати різні варіанти розвитку подій.
- Діаграми пакетів: дозволяють організувати модель системи на рівні пакетів, групуючи схожі елементи та спрощуючи управління складними структурами. Вони сприяють впорядкуванню великої кількості класів та інших елементів системи.
- Діаграми комунікації: моделюють взаємодію між об'єктами за допомогою потоків повідомлень, що показують послідовність та контекст взаємодії, роблячи акцент на комунікацію між учасниками процесу.
- Діаграми композиції: застосовуються для моделювання складних структур, що складаються з багатьох компонентів та їх взаємозв'язків. Вони дозволяють демонструвати цілісність великих модулів у системі та їх взаємодію між собою.
- Діаграми розробки: описують процес розробки програмного забезпечення, включаючи різні етапи, ролі, завдання та залежності між ними. Це допомагає побудувати стратегію розробки, яка враховує ресурси, етапи тестування та релізу.

Кожен тип діаграм UML надає цінну інформацію для різних етапів проєкту, починаючи від аналізу вимог до системи і закінчуючи її впровадженням та підтримкою. Завдяки цим діаграмам команди проєкту

можуть краще зрозуміти вимоги, виявити потенційні проблеми на ранніх етапах, забезпечити узгодженість між частинами системи та ефективно передавати інформацію між усіма учасниками проєкту [16].

Діаграма UML для даної дипломної роботи представлена нижче (рис. 2.1), що відображає конкретний підхід до проектування та дозволяє структурувати інформацію для наступних етапів реалізації.

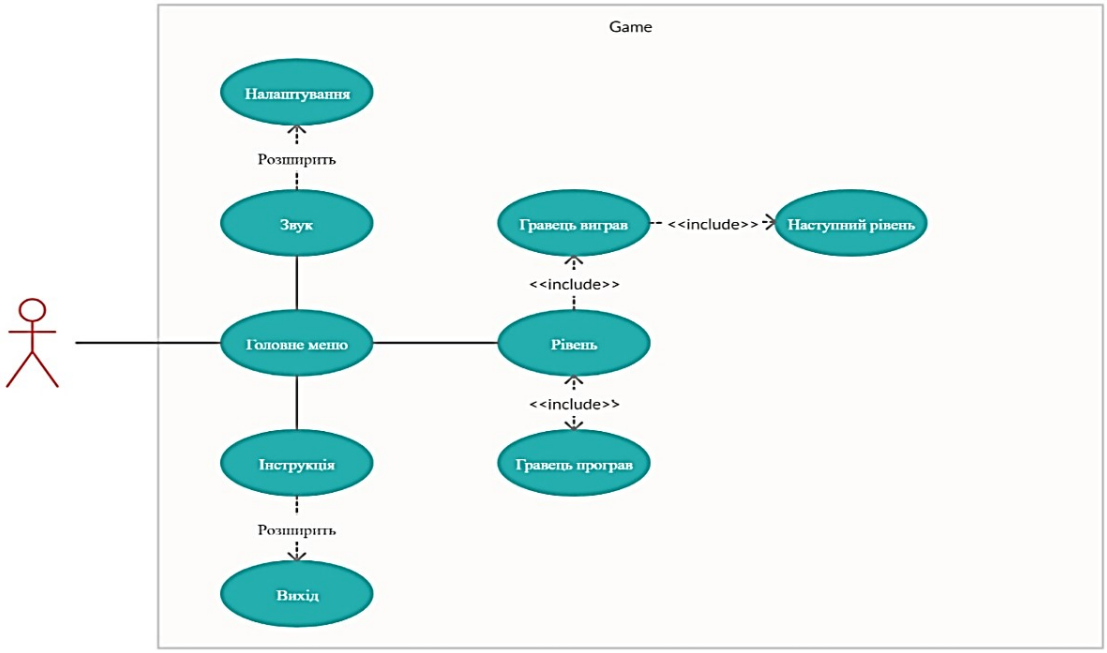


Рис. 2.1. Діаграма можливостей та варіантів використання.

РОЗДІЛ 3. КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Вибір мови програмування та середовища розробки

Для розробки цієї комп'ютерної гри обрана мова Python завдяки її легкості, доступності та багатому набору інструментів для створення ігрових проєктів. Python популярна серед розробників завдяки широкому вибору бібліотек і фреймворків, які спрощують процес створення ігор та розширюють їх можливості [17].

Одним із ключових інструментів став Pygame — спеціалізована бібліотека для розробки ігор, яка надає потужні інструменти для роботи з графікою, звуком, анімацією та фізикою. Вона дозволяє ефективно реалізувати ігрові механіки, підтримує різні формати зображень і аудіо, що робить процес створення інтерактивних елементів зручнішим та швидшим. Для створення ігрових карт та візуалізації рівнів використовується графічний редактор Tiled. Цей редактор допомагає структурувати ігровий простір, дозволяючи легко налаштовувати об’єкти, тайли та колізії.

Комбінація Python, Pygame та Tiled створює гнучку та ефективну платформу для розробки сучасних комп’ютерних ігор із зручним інтерфейсом та великою кількістю можливостей для персоналізації та адаптації.

Мова програмування Python, розроблена Гвідо ван Россумом у 1990 році, є інтерпретованою, об’єктно-орієнтованою мовою високого рівня зі строгою динамічною типізацією. Вона має багато переваг для розробників, таких як підтримка складних структур даних, динамічна семантика та гнучке зв’язування, що робить Python чудовим інструментом для швидкого розгортання проєктів. Підтримка модулів і пакетів забезпечує модульність і повторне використання коду, що полегшує розробку складних проєктів та забезпечує його універсальність на різних платформах [18].

Основні бібліотеки та інструменти:

- Pygame – спеціалізована бібліотека для створення ігор, що надає потужні інструменти для роботи з графікою, звуком, анімацією та фізикою, що робить процес розробки простішим та ефективнішим.
- Tiled – графічний редактор, який використовується для створення тайлованих карт ігрових рівнів, що дозволяє організовувати об’єкти та налаштовувати колізії у зручному інтерфейсі .

Python підтримує кілька парадигм програмування: об'єктно-орієнтовану, процедурну, функціональну та аспектно-орієнтовану, що дає розробникам гнучкість у виборі підходу до вирішення різних завдань проєкту [5].

Переваги Python для розробки ігор:

- Лаконічний синтаксис: Python має простий та зрозумілий синтаксис, що значно полегшує читання і написання коду. Відступи використовуються для структуризації блоків коду, що сприяє покращенню його читабельності.
- Переносність: Python — інтерпретована мова, що дозволяє виконувати код на різних платформах без модифікацій, спрощуючи розгортання проєктів на різних операційних системах.
- Масштабований стандартний дистрибутив: Python надає великий набір вбудованих модулів для виконання різноманітних завдань, включно з підтримкою графічних інтерфейсів, що полегшує створення інтерактивних програм.
- Інтерактивний режим: Python дозволяє виконувати команди в інтерактивному режимі, що є особливо корисним для швидкого тестування, налагодження та розв’язання невеликих задач [1].
- Зручне середовище розробки: Python має стандартне середовище IDLE, яке надає простий редактор коду з базовими інструментами для написання, налагодження та запуску програм.
- Математичний функціонал: Python надає потужні вбудовані інструменти для виконання математичних операцій. Мова дозволяє

працювати з комплексними числами, обробляти великі цілі числа та виконувати розрахунки різної складності, що фактично перетворює Python на багатофункціональний науковий калькулятор [21].

- Python з відкритим кодом: Мова Python розроблена з відкритим вихідним кодом, що дозволяє програмістам вільно використовувати, змінювати та адаптувати її для вирішення специфічних завдань. Це підтримує активну спільноту розробників, які постійно працюють над вдосконаленням мови, створюють нові модулі та розширюють її можливості. Завдяки цьому Python продовжує розвиватися та вдосконалюватися, роблячи її ще більш потужним інструментом для розробки програмного забезпечення.
- Забезпечення безпеки та стабільності: Відкритий код Python дозволяє будь-якому користувачеві переглядати, перевіряти та вдосконалювати його, що сприяє підвищенню безпеки програм, розроблених на цій мові. Відкрита модель також дозволяє ефективно усувати потенційні помилки та вразливості, що забезпечує стабільність і надійність програмного забезпечення.
- Підтримка та зростання популярності: Оскільки Python є мовою з відкритим кодом, вона швидко поширюється серед розробників завдяки активній спільноті, яка сприяє її розвитку, створює нові інструменти та бібліотеки. Це робить Python затребуваною мовою в широкому спектрі галузей і гарантує її подальше зростання і розвиток [24].
- Високорівневі структури даних: Python пропонує ефективні високорівневі структури даних, що полегшує обробку інформації та маніпуляцію даними. Це дозволяє розробникам створювати складніші додатки з меншими зусиллями, оптимізуючи час на розробку.
- Простий та зручний синтаксис: Синтаксис Python вирізняється своєю простотою, що спрощує процес написання коду. Використання відступів для структурування коду значно покращує читабельність, що робить Python чудовим вибором для швидкої розробки програмних додатків, як для початківців, так і для досвідчених розробників.

- Широкий спектр застосування: Завдяки своїй гнучкості та доступності, Python став однією з найпоширеніших мов програмування, яку використовують у веб-розробці, наукових дослідженнях, штучному інтелекті, аналізі даних та багатьох інших сферах. Велика кількість бібліотек та фреймворків робить Python універсальним інструментом для вирішення різноманітних програмних завдань [23].
- Pygame — бібліотека для розробки ігор: Pygame є потужним набором крос-платформових модулів, спеціально розроблених для створення ігор на Python. Ця бібліотека надає широкий функціонал для роботи з графікою, звуком, управлінням введенням користувача та іншими аспектами ігрового процесу, що спрощує процес створення ігор для розробників усіх рівнів.
- Відкритість та співпраця: Pygame, розроблений як альтернатива pySDL, пропонує розробникам доступ до вихідного коду за ліцензією GNU Lesser General Public License, що надає свободу використання та модифікації. Така відкритість сприяє розвитку ігрової спільноти, дозволяючи розробникам ділитися своїми проектами, створювати нові ігри та вдосконалювати вже існуючі.
- Зручність та функціональність для ігор: Завдяки Pygame, розробники можуть легко керувати графікою, анімацією, обробкою звуку та обробкою введення, що надає всі необхідні інструменти для створення захоплюючих ігор на різних платформах [22].

Таким чином, використання Python разом із Pygame та Tiled дає змогу ефективно створювати ігри з багатим функціоналом та зручним процесом розробки. Python забезпечує простий синтаксис, вбудовану підтримку різних бібліотек та гнучкість у програмуванні, що робить його ідеальним вибором для реалізації даного проєкту. Технічне завдання на програмне забезпечення можно переглянути в таблиці 3.1.

Таблиця 3.1

Технічне завдання

Розділ	Підрозділ	Опис
Вступ	Назва програми	MYSTERY of MANSION
	Категорія	Патформер
Вимоги до ПЗ	Функціональні вимоги	➢ - Оптимізація для роботи на слабких пристроях ➢ - Тестування на стійкість до збоїв
	Нефункціональні вимоги	➢ Різноманітність типів ворогів ➢ - Наявність ігрових рівнів
	Вимоги до обладнання	Сумісність з Windows, Linux, OS X, Android, iOS
Стадії та етапи розробки	Стадії та етапи розробки	Подано в таблиці 3.1
Умови використання	Тип обслуговування	Спеціального обслуговування не потребує, окрім можливих оновлень.
	Потреби у персоналі	Робота гри не потребує обслуговуючого персоналу
	Вимоги до коду та мов програмування	Використання мови Python для розробки коду та програми Tiled для створення рівнів
	Вимоги до захисту данних	Дотримання вимог закону України «Про захист персональних даних»
Економічні вимоги	Ефективність	Середній рівань
	Цінність для стратегії	Низький рівень
Контроль якості	Види тестування ПЗ	1. Функціональне тестування: - 1.1 Модульне тестування на етапі розробки - 1.2 Інтеграційне тестування по завершенні розробки - 1.3 Приймальне тестування
	Загальні вимоги до перевірок	2.1 Нефункціональне тестування: - 2.2 Тестування встановлення - 2.3 Перевірка зручності інтерфейсу

3.2. Розробка тестів

Тестовий випадок — це основний елемент процесу тестування, який представляє собою конкретну послідовність кроків, умови та параметри, необхідні для перевірки правильності роботи певної функції або її частини. Його головна мета — переконатися, що функціонал працює згідно з заданими вимогами та очікуваннями, а також поводиться належним чином у різноманітних умовах і ситуаціях, що можуть виникнути під час його використання. Тестовий випадок включає в себе набір критеріїв, які дозволяють визначити, чи є результат виконання функції правильним і чи відповідає він вимогам [25].

Тестові випадки можуть бути поділені на два основні типи: позитивні та негативні.

Позитивні тестові випадки застосовують коректні, правильні дані та передбачають перевірку того, чи функція виконується так, як це передбачено в нормальних умовах використання. Вони дозволяють верифікувати правильність реалізації алгоритмів і механізмів, що стоять за функцією, та перевірити, чи отримує користувач або система очікуваний результат за умови, що всі вхідні дані відповідають стандартам.

Негативні тестові випадки, у свою чергу, орієнтовані на перевірку реакції функції на некоректні або неочікувані умови. Вони включають використання помилкових або неіснуючих даних, а також дослідження того, як система поводиться у випадках, коли виникають помилки, такі як недоступність бази даних, помилки в запитах до серверів або переповнення пам'яті. Негативні тести спрямовані на перевірку того, чи здатна функція належно обробляти неочікувані або критичні ситуації, запобігаючи збоям та помилкам у роботі програми [30].

Поєднання позитивних і негативних тестових випадків забезпечує комплексний підхід до тестування, що дозволяє не лише підтвердити правильність роботи функції в звичайних умовах, але й гарантувати її

стабільність і безпеку під час роботи з помилковими або некоректними даними. Використовуючи позитивні тестові випадки, розробники можуть переконатися, що система працює так, як вона повинна, а за допомогою негативних випадків виявити потенційні проблеми та усунути їх до того, як система потрапить в реальне середовище експлуатації [26].

Всі тестові кейси детально описані в таблицях 3.2–3.9, що дозволяє систематизувати процес тестування.

Таблиця 3.2

Тест-кейс "Вхід в гру"

Назва:	**Вхід в гру**	
Функція:	Перевірка запуску гри.	
Дія	Очікуваний результат	Результат тесту: пройдено/ не пройдено
Передумова:		
Кроки тесту:		
Війти в додаток	Запуск основного меню	Пройдено
Стартова локація	Відображення персонажу на початковій локації	Пройдено
Натиснути на кнопку «Вхід»	Запуск гри	Пройдено

Таблиця 3.3

Тест-кейс «Переміщення персонажу»

Назва:	**Переміщення персонажа**	
Функція:	Перевірка можливості переміщення аватару.	
Дія	Очікуваний результат	Результат тесту: пройдено/ не пройдено
Передумова:		
Війти в додаток	Запуск основного меню	Пройдено
Натиснути на кнопку «Вхід»	Запуск гри	Пройдено
Кроки тесту:		
Натискаємо кнопку A (англійська клавіатура)	Персонаж переміщується вліво	Пройдено
Натискаємо кнопку D (англійська клавіатура)	Персонаж переміщується вправо	Пройдено
Натискаємо кнопку “ ”	Персонаж підстрибує	Пройдено
Натискаємо кнопку “ ” кілька разів поспіль	Персонаж підстрибує лише раз. За умови наявності опори для стрибку..	Пройдено

Таблиця 3.4

Тест-кейс «Підбір зірочок»

Назва:	**Підбір зірок**	
Функція:	Перевірка коректності роботи ігрових об’єктів – зірок	
Дія	Очікуваний результат	Результат тесту: пройдено/ не пройдено
Передумова:		
Війти в додаток	Запуск основного меню	Пройдено
Натиснути на кнопку «Вхід»	Запуск гри	Пройдено
Кроки тесту:		
Рухаючи аватар, шукаємо зірку. Наводимо персонажа на неї .	Ігровий об'єкт зірка зникає, в інвентарі змінюється кількість зібраних зірок на +1 одиницю .	Пройдено

Таблиця 3.5

Тест-кейс «Поранення»

Назва:	**Поранення**	
Функція:	Перевірка зменшення ОЗ аватару.	
Дія	Очікуваний результат	Результат тесту: пройдено/ не пройдено
Передумова:		
Війти в додаток	Запуск основного меню	Пройдено
Натиснути на кнопку «Вхід»	Запуск гри	Пройдено
Кроки тесту:		
Рухаючи аватар знаходимо перешкоду (монстра чи пастку).Об’єднуємо координати аватара та перешкоди..	У лівому верхньому куту екрану шкала здоров’я починає зменшуватись.	Пройдено

Таблиця 3.6

Тест-кейс «Втрата життя»

Назва:	**Втрата життя**	
Функція:	Перевірка коректності смерті персонажа	
Дія	Очікуваний результат	Результат тесту: пройдено
Передумова:		
Війти в додаток	Запуск основного меню	Пройдено
Натиснути на кнопку «Вхід»	Запуск гри	Пройдено
Кроки тесту:		

Продовження таблиці 3.6

Переміщуючи аватар знаходими монстра або перешкоду та наводимо аватар на неї.	Кількість життів зменшується на -1.	Пройдено
Дія	Очікуваний результат	Результат тесту: пройдено
Постумова:		

Таблиця 3.7

Тест-кейс «Закінчення гри через смерть»

Назва:	**Смерть**	
Функція:	Перевірка можливості закінчення гри при умові закінчення життів.	
Дія	Очікуваний результат	Результат тесту: пройдено
Передумова:		
Війти в додаток	Запуск основного меню	Пройдено
Натиснути на кнопку «Вхід»	Запуск гри	Пройдено
Кроки тесту:		
Переміщуючи персонажа, шукаємо монстра. Наводимо персонажа на нього. Повторюємо поки кількість життів не буде рівнятись 0.	Гра завершається чорним екраном з написом “Ви програли”/”Game Over”.	Пройдено
Дія	Очікуваний результат	Результат тесту: пройдено
Постумова:		
Натискаємо Space	Ми опиняємося в головному меню.	Пройдено

Таблиця 3.8

Тест-кейс «Збір ключів»

Назва:	**Ключ**	
Функція:	Перевірка коректності нарахування балів за вбивство монстрів	
Дія	Очікуваний результат	Результат тесту: пройдено
Передумова:		
Війти в додаток	Запуск основного меню	Пройдено
Натиснути на кнопку «Вхід»	Запуск гри	Пройдено
Кроки тесту:		
Переміщуючи аватар, шукаємо ігровий предмет “ключ”. Підбираємо його.	Ключ зникає	Пройдено

Продовження таблиці 3.8

Дія	Очікуваний результат	Результат тесту: пройдено
Відкриваємо інвентар.	Навпроти підібраного ключа має бути напис “знайдено”	Провалений

Таблиця 3.9

Тест-кейс «Перехід на наступну локацію»

Назва:	**Перехід на наступну локацію**	
Функція:	Перевірка переходу	
Дія	Очікуваний результат	Результат тесту:
Передумова:		
Війти в додаток	Запуск основного меню	Пройдено
Натиснути на кнопку «Вхід»	Запуск гри	Пройдено
Кроки тесту:		
Проходимо локацію знаходими ключ та відповідні йому двері.	Використовуємо ключ відповідною клавішею.	Пройдено
Чекаємо	З’являється наступна локація.	Провалено

3.3. Виконання тестів

У таблицях 3.10 та 3.11 описані Баг-репорти на негативні тест-кейси.

Таблиця 3.10

Баг-репорт «Бали за монстрів»

Короткий опис	Після підбору ключа не зьявляється відповідний запис в інвентарі про його знаходження.
Проект	Гра-платформер на MYSTERY of MANSION
Компонент додатку	Результати
Номер версії	—
Важливість:	S3 Незначна (Minor)
Пріоритет:	P2 Середній (Medium)
Статус	Відкритий
Автор	Кулик Роман
Призначений на виправлення	Кулик Роман
Кроки відтворення	Переміщуючись по локації шукаємо ключ , підбираємо, преходимо до інвентаря.
Фактичний результат	Навпроти відповідного ключа зтявляється дубльована помітка знайдено.
Очікуваний результат	Навпроти відповідного ключа має відобразитися “знайдено”

Таблиця 3.11

Баг-репорт «Перехід до наступної локації»

Короткий опис	Ключ від локації працює проте завантаження а саме перехід до неї не відбувається..
Проект	Гра-платформер на Pyton «MYSTERY of MANSION»
Компонент додатку	Гра
Номер версії	—
Важливість:	S1 Блокуюча (Blocker)
Пріоритет:	P1 Високий (High)
Статус	Закрито
Автор	Кулик Роман
Призначений на виправлення	Кулик Роман
Кроки відтворення	Проходимо рівень, шукаємо ключ та використовуємо його на відповідному переході.
Фактичний результат	Через 1-2 секунд очікування завантажується нова локація
Очікуваний результат	Повністю співпадає з фактичним.

3.4. Інструкція користувача до гри "MYSTERY of MANSION"

У цій роботі розроблено програмний продукт під назвою "MYSTERY of MANSION" — динамічну гру-платформер. Її основна мета полягає в розважанні користувачів, водночас сприяючи розвитку їх координації, швидкості реакції, а також логічного мислення. Гра може слугувати як захопливий інструмент для тренування уважності й спритності, так і для приємного проведення часу. "MYSTERY of MANSION" доступна для запуску на персональних комп'ютерах із різними характеристиками, що робить її універсальною для широкого кола користувачів [19].

Системні вимоги

Для роботи гри "MYSTERY of MANSION" необхідно мати:

- Персональний комп'ютер (ноутбук або стаціонарний).
- Встановлену операційну систему (Windows, macOS або Linux).

Перед встановленням необхідно, щоб на вашому комп'ютері вже був встановлений Python версії 3.10+.

Рекомендується Python 3.11+

для Windows:

- Скопіюйте репозиторій на свій комп'ютер
- Створіть віртуальне середовище: python -m venv venv
- Активуйте віртуальне середовище:venv/Scripts/activate
- Встановіть зовнішні бібліотеки, виконавши: pip install -r requirements.txt
- Запустіть файл main.py

для Linux/Ubuntu:

- Скопіюйте репозиторій на свій комп'ютер
- Створіть віртуальне середовище: python3 -m venv env
- Активуйте віртуальне середовище: source env/bin/activate
- Встановіть зовнішні бібліотеки, виконавши: pip install -r requirements.txt
- Запустіть файл main.py

Головне меню

При запуску програми на екрані з’являється головне меню (рис. 3.1). Воно має простий і зручний інтерфейс, що дозволяє користувачеві обрати необхідну дію. Основу коду якого можно переглянути в додатках (A,C,D).

Рис. 3.1. Головне меню

Кнопка "NEW GAME" переносить гравця до ігрового процесу, де починається перший рівень (рис. 3.2). Уривок коду якого продемонстровано у додатку (В).

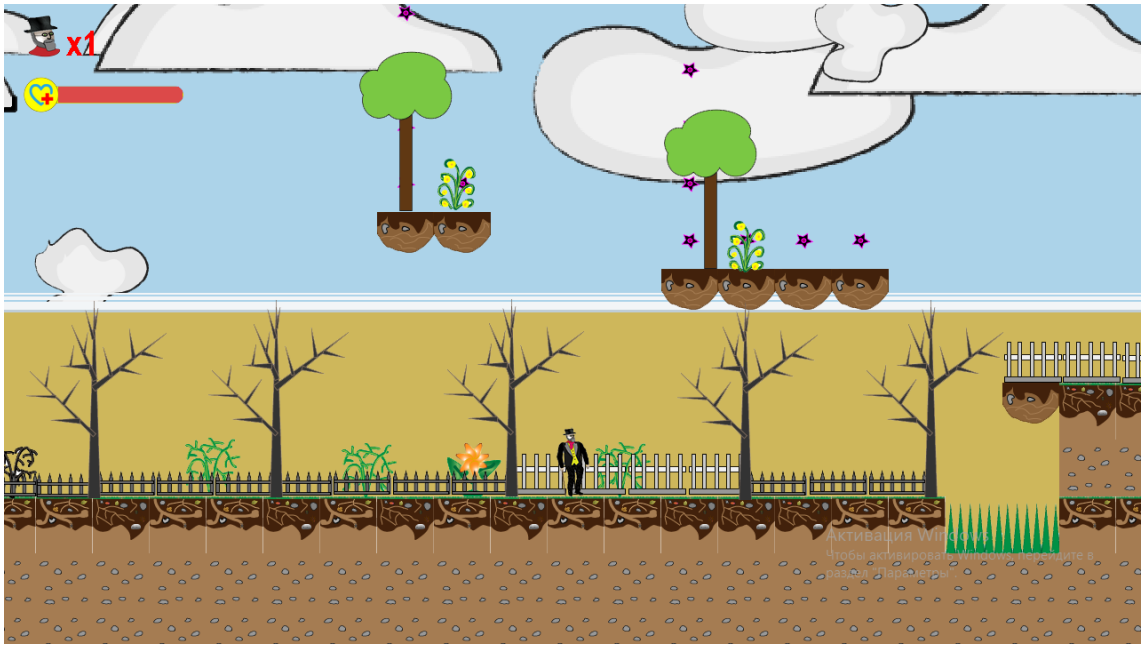


Рис. 3.2. Стартова локація

Управління

Для керування персонажем передбачені наступні клавіші:

- A (або стрілка "ліворуч") — пересування вліво.
- D (або стрілка "праворуч") — пересування вправо.
- SPASE — стрибок персонажа.

Управління інтуїтивне і підходить навіть для новачків, а також адаптоване для зручної гри як на клавіатурі ноутбука, так і на зовнішній клавіатурі.

Опис ігрового процесу

Ігровий процес складається з кількох рівнів, кожен із яких має свою структуру, перешкоди та ворогів. Завдання гравця — подолати всі перешкоди та дістатися до фіналу рівня [27].

Монстри

У грі зустрічаються різні види ворогів:

1. Перший вид монстрів (рис. 3.3):
   - Рухаються по горизонталі.
   - Можуть бути знищені, якщо стрибнути їм на голову.

Рис. 3.3. Перший тип монстрів

2. Другий вид монстрів (рис. 3.4):
   - Рухаються вправо-вліво.
   - Незнищенні при стрибку на голову, тому їх краще уникати.

Кожен ворог додає грі динамічності та змушує гравця обдумувати свої дії.

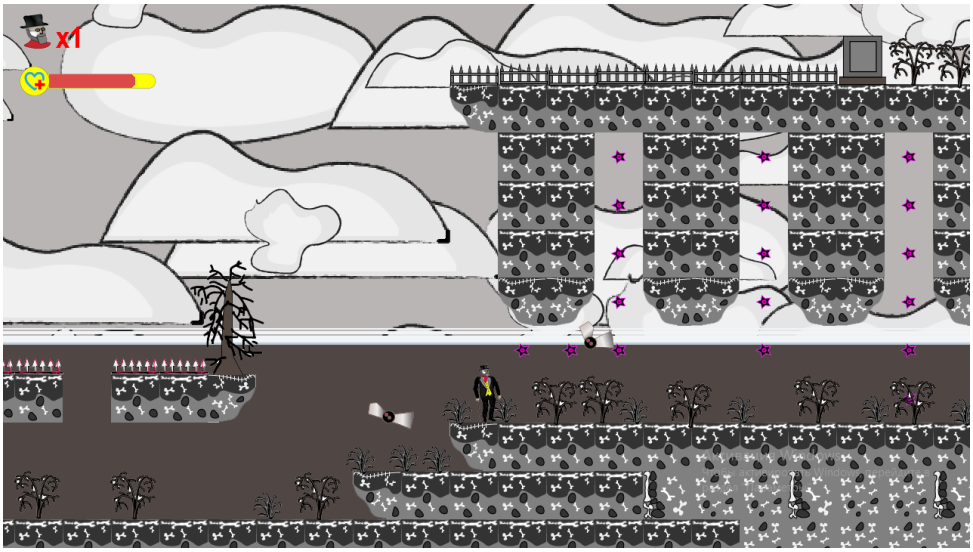


Рис. 3.4. Другий тип монстрів

Життя персонажа

Персонаж має обмежену кількість життів. У разі поразки:

- Персонаж втрачає одне життя.
- Коли втрачаєш останне життя гра завершається (рис. 3.5).

Гравець має можливість повторити спробу, поки не досягне успіху або поки не закінчаться життя [29].

Рис. 3.5.Заверешнння гри через втрату всіх життів.

Додаткові функції

Для повернення в головне меню необхідно натиснути Space, а після появи темного екрану — клавішу ESC.

Додаткові можливості гри

- Інтерактивне навчання: у грі передбачено підказки на перших рівнях, які допомагають новачкам освоїти основи управління.
- Збереження прогресу: гра автоматично запам’ятовує досягнення гравця, що дозволяє продовжити гру з місця, на якому зупинилися.
- Оптимізація для різних пристроїв: "MYSTERY of MANSION" працює плавно навіть на комп'ютерах із мінімальними характеристиками.
- Налаштування звукового та мовного супровіду (рис 3.6).

- А також в грі реалізований власний генератор рівнів (рис 3.7), де за бажанням гравець може створити власний або відредагувати наявний шаблон і урізноманітнити гру [33]. Частини коду надані в додатках (E,F).

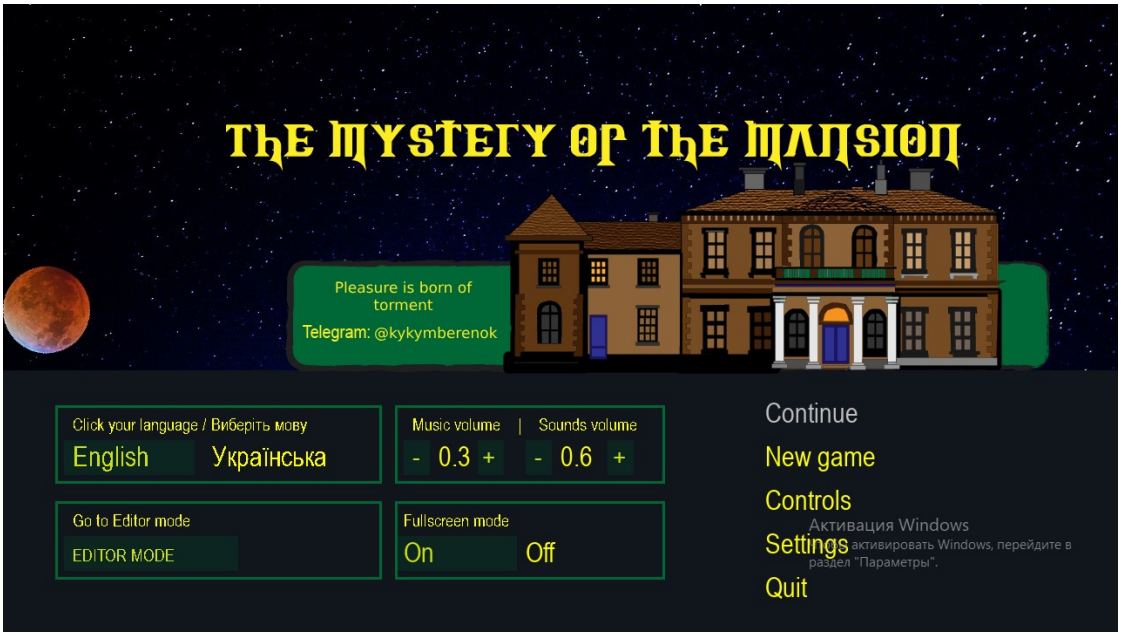


Рис. 3.6 Налаштування.

Гра "MYSTERY of MANSION" створена, щоб забезпечити користувачам не лише захопливий ігровий процес, але й можливість вдосконалювати свої навички в ігровій формі.

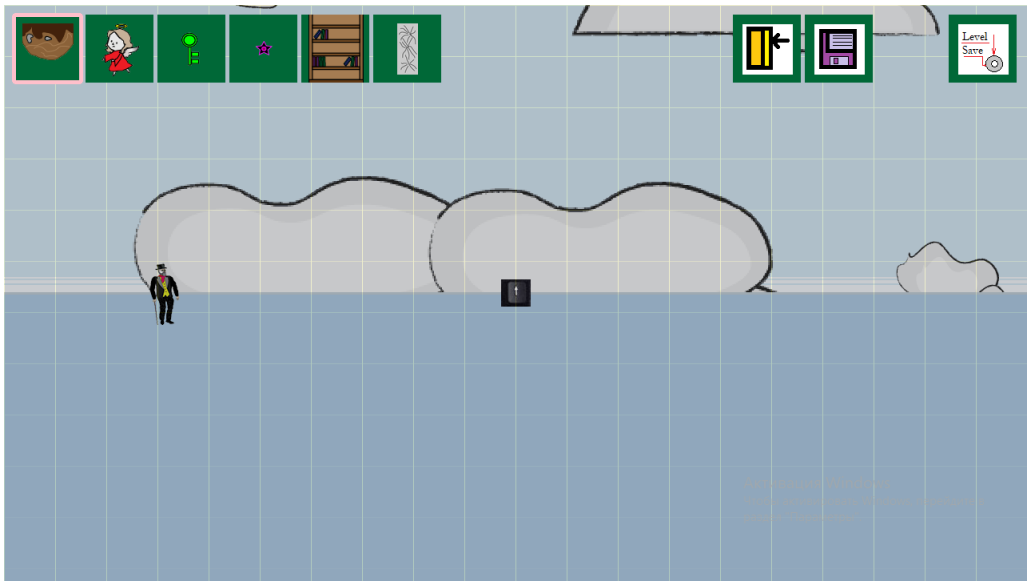


Рис. 3.7 Редактор рівнів.

РОЗДІЛ 4. ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ РОЗРОБЛЕННЯ ПРОГРАМНОГО ПРОДУКТУ

4.1. Визначення трудомісткості розробки програмного забезпечення

Оцінювання трудових витрат під час створення програмного забезпечення є важливим, але складним процесом через творчий характер роботи програміста. Визначення трудомісткості здійснюється із використанням різних модельних систем, що дозволяють досягти різного рівня точності.

Трудомісткість вказує на час, необхідний для виконання певної функціональності або технологічного етапу. Це ключовий показник, що характеризує технологічність програмного продукту. Скорочення трудомісткості є важливим завданням, яке сприяє зниженню витрат і підвищенню ефективності розробки.

Для оцінки трудомісткості використовується формула:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \quad (4.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі;

$t_и$ – витрати праці на дослідження алгоритму розв’язання задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

$t_п$ – витрати праці на програмування за готовою блок-схемою;

$t_{отл}$ – витрати праці на налагодження програми;

$t_д$ – витрати праці на підготовку документації.

За проведеними розрахунками загальна трудомісткість становить:

$$t = 350 + 15 + 40 + 15 + 180 + 120 = 720 \text{ людино-годин}$$

Витрати визначаються залежно від кількості операторів у програмному

продукті. Умовну кількість операторів можна обчислити за формулою:

$$Q = q \cdot C \cdot (1 + p), \tag{4.2}$$

де:

- q — очікувана кількість операторів;
- C — коефіцієнт складності програми (зазвичай від 0,9 до 1,2);
- p — коефіцієнт кореляції програми під час розробки (0,05–0,1).

Для поточного проекту:

$$Q = 445 \cdot 0{,}9 \cdot (1 + 0{,}1) = 490{,}05 \text{ людино-годин}$$

Витрати праці на дослідження алгоритму задачі t_u визначаються з урахуванням уточнення опису задачі та кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k} \text{ людино-годин}, \tag{4.3}$$

де

- B — коефіцієнт збільшення витрат через недостатнє опрацювання опису задачі (від 1,2 до 1,5);
- k — коефіцієнт кваліфікації програміста, залежний від його досвіду.

Коефіцієнт B відображає якість початкового опису задачі та необхідність його уточнення. У реальних умовах цей коефіцієнт залежить від складності завдання й варіюється в межах 1,2–1,5.

Коефіцієнт кваліфікації k демонструє рівень підготовки виконавця. Його значення залежить від стажу роботи та рівня знань.

Поглиблення аналізу таких коефіцієнтів і параметрів сприяє оптимізації процесу розробки, дозволяючи точніше планувати ресурси й строки виконання.[41]

Рівень кваліфікації працівників

Коефіцієнт кваліфікації k залежить від стажу працівників і використовується для коригування витрат праці:

- До 2 років: k=0,8
- 2–3 роки: k=1,0
- 3–5 років: k=1,1–1,2
- 5–7 років: k=1,3–1,4
- Понад 7 років: k=1,5–1,6

Для даного проєкту працівники мають стаж від 2 до 3 років, тому k=1

Витрати праці на дослідження алгоритму

Формула для розрахунку:

$$t_u = \frac{490 \cdot 1,4}{75 \cdot 1} = 9,14 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20K25) \cdot k} \text{ людино-годин} \qquad (4.4)$$

Розрахуємо для нашого програмного продукту:

$$t_a = \frac{490}{20 \cdot 1} = 24,5 \text{ людини-годин}$$

Витрати часу на складання програми за готовою блок-схемою:

$$t_n = \frac{Q}{(20K25) \cdot k} \text{ людино-годин} \qquad (4.5)$$

Розрахуємо витрати часу на складання програми за готовою блок-схемою:

$$t_n = \frac{490}{21 \cdot 1,3} = 23,3 \text{ людино-годин}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5) \cdot k} \text{ людино-годин,} \quad (4.6)$$

Витрати праці на автономне налагодження одного завдання складає:

$$t_{отл} = \frac{720}{4 \cdot 1} = 180 \text{ людино-годин}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до} \text{ людино-годин,} \quad (4.7)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису

$$t_{др} = \frac{Q}{(15..20) \cdot k} \text{ людино-годин} \quad (4.8)$$

$t_{до}$ – трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0,75 \cdot t_{др} \text{ людино-годин} \quad (4.9)$$

Розрахуємо трудомісткість підготовки матеріалів і рукопису та редагування, друку та оформлення документації:

$$t_{др} = \frac{490,}{15 \cdot 1} = 32,6 \text{ людино-годин}$$

$$t_{до} = 0{,}75 \cdot 25{,}13 = 24{,}5 \text{ людино-годин.}$$

Тоді витрати праці на підготовку документації будуть складати

$$t_{д} = 32{,}6 + 24{,}5 = 57{,}1 \text{людино-годин.}$$

Загальна трудомісткість

Підсумуємо всі складові:

$$t = tu + ta + tn + totл + tд.$$

$$t=9{,}14+24{,}05+23{,}3+180+57{,}1=294{,}04 \text{ людино-годин.}$$

Отримані результати демонструють розподіл витрат праці за основними етапами розробки програмного продукту, що сприяє точному плануванню ресурсів і оптимізації процесу.

4.2. Витрати на створення програмного забезпечення

Розрахунок витрат на створення програмного забезпечення включає основну та додаткову заробітну плату, а також інші витрати, пов’язані із трудовою діяльністю.[42]

Основна заробітна плата

Основна заробітна плата є винагородою за виконання роботи згідно з установленими нормами праці. Вона визначається на основі тарифних ставок, посадових окладів або відрядних розцінок і є базовим елементом системи оплати праці.

Додаткова заробітна плата

Додаткова заробітна плата включає оплату за:

- Працю понад встановлену норму.

- Досягнення високих результатів у роботі.
- Особливі умови праці.

Складовими цієї частини заробітної плати є доплати, надбавки та компенсаційні виплати.

Доплати та надбавки

Доплати й надбавки є окремими елементами структури заробітної плати та частиною тарифної системи. Ці виплати забезпечують диференціацію заробітної плати залежно від:[43]

- Кваліфікації працівника.
- Складності виконуваних завдань.
- Рівня відповідальності.
- Умов і інтенсивності праці.
- Специфіки підприємства.

Тарифна система

Тарифна система є сукупністю державних нормативів, що регулюють оплату праці. Вона базується на таких складових:

- Тарифні ставки — визначають оплату праці залежно від розряду.
- Тарифні сітки — встановлюють співвідношення оплати між різними категоріями працівників.
- Схеми посадових окладів — розраховують оплату для посад різного рівня складності.
- Тарифно-кваліфікаційні характеристики — визначають вимоги до рівня знань і практичних навичок для виконання робіт.

Тарифні ставки можуть бути скориговані для працівників, які працюють у шкідливих або небезпечних умовах, що передбачає додаткову оплату.

Тарифно-кваліфікаційні довідники

Тарифно-кваліфікаційні довідники — це систематизовані переліки робіт і професій, що визначають рівень кваліфікації та є основою для уніфікації оплати праці в різних галузях.

Розрахунок робочих годин у році

Число робочих годин у році залежить від тривалості робочого дня, кількості робочих днів на рік, а також офіційних святкових і вихідних днів. Формула для розрахунку:

$$n(р) = (N - N(n) - N(в)) \cdot 8, \quad (4.10)$$

де N – загальне число днів у році,

N(п) – число святкових днів у році,

N(в) – число вихідних днів у році.

Число святкових днів у році – 10, а вихідних – 105.

Отже, число робочих годин у році дорівнює:

$$n(р) = (365 - 10 - 105)) \cdot 8 = 2000(год).$$

Середня оплата за годину програміста знаходиться за співвідношенням:

$$С_{розр.} = \frac{ФЗР_{СН}}{n(р)}, \quad (4.11)$$

де $ФЗР_{сн}$ – річний фонд грошового забезпечення.

Для нашого проекту:

$$С_{розр.} = \frac{120000}{2000} = 60{,}00(грн).$$

Отже, витрати по основній оплаті праці ($З_{осн.}$) розробників програми складають:

$$З_{осн.} = 720 \cdot 60{,}00 = 43\ 200(грн).$$

Для визначення загальної суми на оплату праці необхідно додати

надбавки [44]. Приймаємо розмір додаткової заробітної плати 13% від основної заробітної плати:

$$З_{доп.} = 43200 \cdot 0{,}15 = 5\,616(грн).$$

Загальний фонд оплати праці:

$$З_{заг.} = 43{,}200 + 5{,}616 = 48\,816(грн).$$

Визначимо розмір єдиного соціального внеску, який складає 22%:

$$ЄСВ = 48816 \cdot 0{,}22 = 10\,739{,}52(грн).$$

$$З_{розр} = 48\,816 + 10\,739{,}52 = 59\,555{,}52(грн).$$

Розробка програмного забезпечення потребує комплексного підходу до планування витрат, які включають не тільки оплату праці розробників, але й вартість використання обладнання, енергоресурсів та інших ресурсів. Витрати поділяються на прямі та непрямі, а також розраховуються для кожного етапу виробництва, щоб забезпечити прозорість та контроль за використанням фінансових ресурсів.

Основна та додаткова заробітна плата

Основна заробітна плата – це винагорода, яку працівники отримують за виконання своїх обов’язків відповідно до встановлених норм праці. Вона визначається на основі тарифних ставок, посадових окладів або відрядних розцінок.

Додаткова заробітна плата нараховується за працю, виконану понаднормово, за особливі умови праці або як заохочення за досягнення високих результатів. Вона включає такі елементи, як доплати за складність робіт, надбавки за кваліфікацію, премії та компенсаційні виплати[45].

Тарифна система та її роль

Тарифна система є основою для визначення рівня оплати праці працівників, виходячи з їхньої кваліфікації, складності виконуваних завдань та умов роботи. Основними елементами тарифної системи є:

- Тарифні ставки – визначають розмір оплати за певний обсяг роботи залежно від розряду працівника.
- Тарифні сітки – регулюють співвідношення в оплаті праці для різних категорій працівників залежно від складності та важливості роботи.
- Тарифно-кваліфікаційні довідники – містять переліки професій та робіт із зазначенням вимог до знань і навичок.

Розрахунок собівартості машинного часу

Для визначення витрат, пов'язаних із використанням ПК під час розробки програмного забезпечення, необхідно розрахувати собівартість однієї години роботи комп'ютера. Собівартість машинного часу залежить від річного фонду корисного часу роботи ПК (за вирахуванням простоїв на профілактику і ремонт) та річних витрат на експлуатацію.[46]

Річний фонд часу роботи ПК визначається за формулою:

$$C_{ПК} = З_{гПК}/T_{гПК} \quad (4.12)$$

Витрати на експлуатацію ПК

Експлуатаційні витрати включають:

1. Амортизаційні відрахування – визначаються на основі балансової вартості ПК і норми амортизації (15% на квартал).
2. Електроенергія – обчислюється з урахуванням потужності ПК, річного фонду часу роботи, вартості електроенергії (1,5 грн за кВт·год) та коефіцієнта інтенсивності використання ПК.
3. Ремонт і профілактика – витрати на ремонт становлять 6% від балансової вартості обладнання.
4. Матеріали та комплектуючі – оцінюються як 2% від балансової

вартості ПК.

5. Непрямі витрати – витрати на експлуатацію, які становлять 5–10% від балансової вартості.

$$Т_{гПК} = 2000 - (6 \cdot 8 + 5 \cdot 12) = 1892(год)).$$

Підсумкова формула річних витрат:

$$З_{гПК} = З_{гАМ} + З_{гЕЛ} + З_{гРЕМ} + З_{гМАТ} + З_{гДР}, \quad (4.13)$$

де $З_{гАМ}$ – відрахування для амортизацію,

$З_{гЭЛ}$ – витрати по електроенергії,

$З_{гРЕМ}$ – витрати на ремонт,

$З_{гМАТ}$ – витрати на заміну комплектуючих,

$З_{гДР}$ – інші витрати.

Сума річних амортизаційних відрахувань визначається за наступною формулою:

$$З_{гАМ} = Ц_{ПК} + НА, \quad (4.14)$$

де $Ц_{ПК}$ – вартість ПК,

НА – дорівнює 15% (у квартал).

Балансова вартість ПК:

$$Ц_{ПК} = Ц_{р} \cdot (1 + К_{ун}), \quad (4.15)$$

де $Ц_{р}$ – ринкова вартість,

$К_{ун}$ – установка й налагодження, рівний 12%.

Розрахуємо суму річних амортизаційних відрахувань і балансову вартість ПК:

$$Ц_{ПК} = 50000 \cdot (1 + 0{,}12) = 56000 \text{ (грн)}$$

$$З_{гАМ} = 56\,000 \cdot 0{,}15 \cdot 4 = 33\,600 \text{(грн)}.$$

Витрати на електроенергію, споживану ПК, визначаються за наступною формулою:

$$З_{гЕЛ} = Р_{чПК} \cdot Т_{гПК} \cdot Ц_{ЕЛ} \cdot А, \quad (4.16)$$

де $Р_{чПК}$ – настановна потужність ПК ($Р_{чПК}$=0,3 (кВт)),

$Т_{гПК}$ – річний фонд корисного часу роботи машини,

$Ц_{ЭЛ}$ – вартість 1 квт/година електроенергії ($Ц_{ЭЛ}$=4,32 (грн)),

А – коефіцієнт інтенсивного використання ПК (0,9-1).

Таким чином, розрахункове значення витрат на електроенергію, споживану ПК, складає:

$$З_{гЕЛ} = 0{,}3 \cdot 1892 \cdot 4{,}32 \cdot 1 = 2452{,}03 \text{(грн)}.$$

Витрати на поточний і профілактичний ремонт приймаються рівними 6% від вартості ПК:

$$З_{гРЕМ} = Ц_{ПК} \cdot 0{,}06 \quad (4.17)$$

Для програмного продукту витрати на поточний і профілактичний ремонт будуть складати:

$$З_{гРЕМ} = 56000 \cdot 0{,}06 = 3360 \text{(грн)}.$$

Витрати на матеріали – витрати необхідні для забезпечення експлуатації ПК приймаються рівними 2% від вартості ПК:

$$З_{гМАТ} = Ц_{ПК} \cdot 2\% \, . \quad (4.18)$$

Після обчислення отримали:

$$З_{гМАТ} = 56000 \cdot 0{,}02 = 1120(грн).$$

Непрямі витрати, тобто витрати пов'язані з експлуатацією ПК приймаються рівними 5-10% вартості ПК:

$$З_{гДР} = Ц_{ПК} \cdot 5\% \quad (4.19)$$

Порахуємо непрямі витрати для проекту:

$$З_{гДР} = 56000 \cdot 0{,}1 = 5\,600(грн).$$

Повні витрати на експлуатацію ПК впродовж року складають:

$$З_{гПК} = 33600 + 2452 + 3360 + 1120 + 5600 = 46132(грн).$$

Собівартість машинного часу ($С_{ПК}$) складає:

$$С_{ПК} = \frac{46132}{1892} = 24{,}38(грн).$$

У ході розробки програмного комплексу машина використовувалася на етапах програмування:

– написання програми за готовою схемою алгоритму;

– налагодження програми на ПК;

– підготовки документації по задачі.

Таким чином, витрати машинного часу склали ($t_{маш}$):

$$t_{маш} = t_{прог} + t_{отл} + t_{док} . \quad (4.20)$$

Порахуємо витрати машинного часу для розробки програмного продукту:

$$t_{маш} = 180 + 57{,}1 + 47{,}35 = 284{,}45 \text{ (чол./год)}.$$

Витрати на оплату машинного часу можна розрахувати за формулою

$$З_{маш} = t_{маш} \cdot С_{ПК}. . \quad (4.21)$$

Розрахуємо витрати на оплату машинного часу:

$$З_{маш} = 284{,}45 \cdot 24{,}38 = 6934{,}89(\text{грн}).$$

Загальні витрати на створення програмного продукту складають:

$$З_{разом} = З_{разр} + З_{маш}. \quad (4.22)$$

Розрахуємо загальні витрати на створення програмного продукту:

$$З_{разом} = 59\,555{,}52 + 6934{,}89 = 66490{,}411(\text{грн}).$$

Кошторис витрат – це структурований план, який об'єднує всі види витрат, пов'язаних із виробничо-фінансовою діяльністю підприємства на визначений період. Він слугує інструментом для управління ресурсами, забезпечуючи прозорість та ефективність фінансових операцій. У кошторисі враховується загальна сума витрат, розподілена за основними категоріями, такими як типи ресурсів, етапи виробничого процесу, рівні управління підприємством, а також специфічні напрямки витрат, що спрямовані на досягнення цілей компанії[47].

Кошторис. витрат на створення програмного продукту відображено в

таблиці 4.1.

Таблиця 4.1

Кошторис. витрат на створення програмного продукту

Стаття витрат		Сума, грн.	Відсоток від загальної суми
1		2	3
ФЗП	$З_{осн}$	43200	40,88%
	$З_{доп}$	5616	5,31%
ЄСВ		10739,52	10,16%
Експлуатаційні витрати	$З_{гАМ}$	33600	31,79%
	$З_{гЭЛ}$	2452,03	2,32%
	$З_{гРЕМ}$	3360	3,18%
	$З_{гМАТ}$	1120	1,06%
	$З_{гДР}$	5600	5,3%
Всього		105 687,55	100%

Джерело: розробка автора на основі [41,42,43,44,45,46,47]

Знайдемо ціну програми:

– прибуток від реалізації 15%;

– податок на вартість 20%

$$Ц_{п} = З_{разом} + П_{р}. \quad (4.23)$$

Після підрахунку:

$$Ц_{п} = 105\,687{,}55 \cdot 1{,}15 = 121540{,}683(грн).$$

Вартість програми з урахуванням ПДВ:

$$Ц_{В} = Ц_{п} + ПДВ \quad (4.24)$$

Після підрахунку:

$$Ц_В = 121540{,}683 \cdot 1{,}2 = 145848{,}82(\text{грн.})$$

4.3. Економічне обґрунтування проекту

Пробний прайс гри виставляємо 80 гривень за 1 копію, середнє число завантажень планується не менше 80-100 разів на місяць.

Термін окупності обчислимо за формулами:

$$О_{програми} = З_{разом}/УЕ_{рік}, \quad (4.25)$$

Тобто, у нашому випадку:

$$О_{програми} = \frac{145848}{80 \cdot 80 \cdot 12} = 1{,}89(\text{роки}).$$

Отже ми показали, що програма окупиться за один рік та дев’ять місяців, після округлення отримуємо два роки, і вже після цього розраховуємо ефект за 3 роки:

$$Е_{n\ роки} = n \cdot УЕ_{рік}, \quad (4.26)$$

де $Е_{n\ роки}$ – ефективність за n років.

$$Е_{3\ роки} = 3 \cdot 76800 = 230400(\text{грн}).$$

Отже можна зробити висновок, що програма буде окуплена за два роки. За три роки її експлуатації вона принесе прибуток в розмірі 84 552 гривень, а економічна ефективність програми за два роки складе 58%.

$$Е_ф = \frac{П_П}{Ц_в} \text{ x } 100 \quad (4.26)$$

$$\text{Ефективність} = \frac{84{,}552}{145848} \text{ x } 100\% = 58\%.$$

ВИСНОВКИ

У межах кваліфікаційної магістерської роботи на тему "Створення 2D платформера з відкритим світом (MYSTERY of MANSION)" було виконано повний цикл розробки програмного продукту — від аналізу вимог до тестування готової гри.

Основні досягнення та результати роботи:

1. Аналіз предметної області:
   - Проаналізовано специфіку жанру 2D платформерів, їх ключові механіки, переваги та популярність серед користувачів.
   - Розглянуто приклади успішних платформерів, що стали орієнтирами для проєктування.
2. Проектування програмного продукту:
   - Розроблено технічне завдання, яке охоплює функціональні, нефункціональні та технічні вимоги до гри.
   - Використано мову UML для побудови діаграм, що ілюструють архітектуру програмного продукту, механізми взаємодії елементів гри та їх поведінку.
3. Розробка гри:
   - Вибрано мову Python та бібліотеку Pygame для реалізації основних ігрових механік: руху персонажа, взаємодії з об’єктами, роботи зі звуком та графікою.
   - Використано редактор Tiled для створення ігрових рівнів, що забезпечило зручність у розробці та інтеграції в код.
   - Створено візуальне оформлення гри в мальованому стилі, що надає їй унікальності.
4. Тестування:
   - Проведено функціональне та нефункціональне тестування, яке включало перевірку запуску гри, переміщення персонажа, взаємодії з ворогами та об'єктами, а також стабільності роботи продукту.

- Результати тестування підтвердили відповідність гри заданим вимогам.

5. Економічне обґрунтування:

- Проведено розрахунок собівартості розробки гри, її потенційної ціни з урахуванням прибутку та ПДВ.
- Визначено термін окупності проєкту — 1,89 роки (після округлення — 2 роки).
- За три роки експлуатації очікується прибуток у розмірі 84,552 грн та економічна ефективність 58%.

6. Практичне значення:

- Розроблений проєкт є мультиплатформенним, що дозволяє використовувати його на різних пристроях (ПК, мобільні телефони, консолі).
- Гра може бути вдосконалена шляхом додавання нових рівнів, функцій, покращення графіки та адаптації під нові платформи.

Розроблена гра "MYSTERY of MANSION" демонструє високий потенціал для подальшого розвитку. Проєкт поєднує в собі простоту використання, привабливий візуальний стиль та інтерактивність, що робить його цікавим для широкої аудиторії. Проведені дослідження та реалізація програмного продукту підтверджують доцільність та перспективність обраної теми.

ПЕРЕЛІК ПОСИЛАНЬ

1. Python.org. “Python Official Documentation.” https://docs.python.org/

2. Pygame.org. “Pygame Documentation.” https://www.pygame.org/docs/

3. Real Python. “Building Games with Python and Pygame.” https://realpython.com/pygame-a-primer/

4. Godot Engine Documentation. “Creating 2D Games.” https://docs.godotengine.org/

5. Game Development with Python. Apress, 2022.

6. Rabin, Steve. “Game AI Pro: Collected Wisdom of Game AI Professionals.” CRC Press, 2013.

7. Tiled Map Editor. “Documentation and Tutorials.” https://doc.mapeditor.org/

8. Brackeys. “How to Make a 2D Platformer Game.” https://brackeys.com/

9. Wenderlich, Ray. “Designing a 2D Platformer Game.” https://www.raywenderlich.com/

10. Unity Documentation. “Platformer Game Design.” https://docs.unity3d.com/

11. Stachniak, Zbigniew. “Platformer Game Programming.” Springer, 2020.

12. Khan Academy. “Intro to Games in Python.” https://www.khanacademy.org/

13. GeeksforGeeks. “Creating a 2D Game with Python.” https://www.geeksforgeeks.org/

14. Audacity Team. “Audacity Tutorial: Editing Audio for Games.” https://manual.audacityteam.org/

15. Steam. “2D Platformer Development.”

https://partner.steamgames.com/

16. O'Reilly Media. “Python Game Programming by Example.” Packt Publishing, 2015.

17. GameDev.net. “2D Platformer Game Mechanics.” https://www.gamedev.net/

18. Packt Publishing. “Hands-On Game Development with Pygame.” 2020.

19. Journal of Game Development. “Level Design in 2D Games.” Taylor & Francis.

20. TutsPlus. “Game Development Tutorials.” https://tutsplus.com/

21. SpringerLink. “Advances in Game Development.” https://link.springer.com/

22. GameFromScratch. “Godot 2D Game Engine Overview.” https://gamefromscratch.com/

23. Aseprite Documentation. “Creating Pixel Art.” https://www.aseprite.org/docs/

24. IndieDB. “Indie Game Development Resources.” https://www.indiedb.com/

25. Sciencedirect. “Game Design and Development.” https://www.sciencedirect.com/

26. Game Design Foundations by Roger Pedersen. Jones & Bartlett Learning, 2003.

27. Gamasutra. “2D Game Design and Development.” https://www.gamasutra.com/

28. FMOD Documentation. “Sound Integration in Games.” https://www.fmod.com/docs/

29. GameMaker Blog. “Developing Platformer Games.” https://www.yoyogames.com/

30. Microsoft Developer Network. “Python for Game Developers.” https://developer.microsoft.com/

31. YouTube. “Python Game Development Tutorials.” https://www.youtube.com/

32. Udemy. “Python for 2D Game Development.” https://www.udemy.com/

33. HarvardX. “Game Development with Python.” edX, https://www.edx.org/

34. MIT OpenCourseWare. “Introduction to Game Development.” https://ocw.mit.edu/

35. Coursera. “Game Design and Development.” https://www.coursera.org/

36. LinkedIn Learning. “Python for Game Development.” https://www.linkedin.com/learning/

37. GitHub. “Open Source Game Projects with Python.” https://github.com/

38. IEEE Xplore. “Advances in Python Game Development.” https://ieeexplore.ieee.org/

39. Medium. “Python Game Development Articles.” https://medium.com/

40. ACM Digital Library. “Game Mechanics and Development.” https://dl.acm.org/

ДОДАТКИ

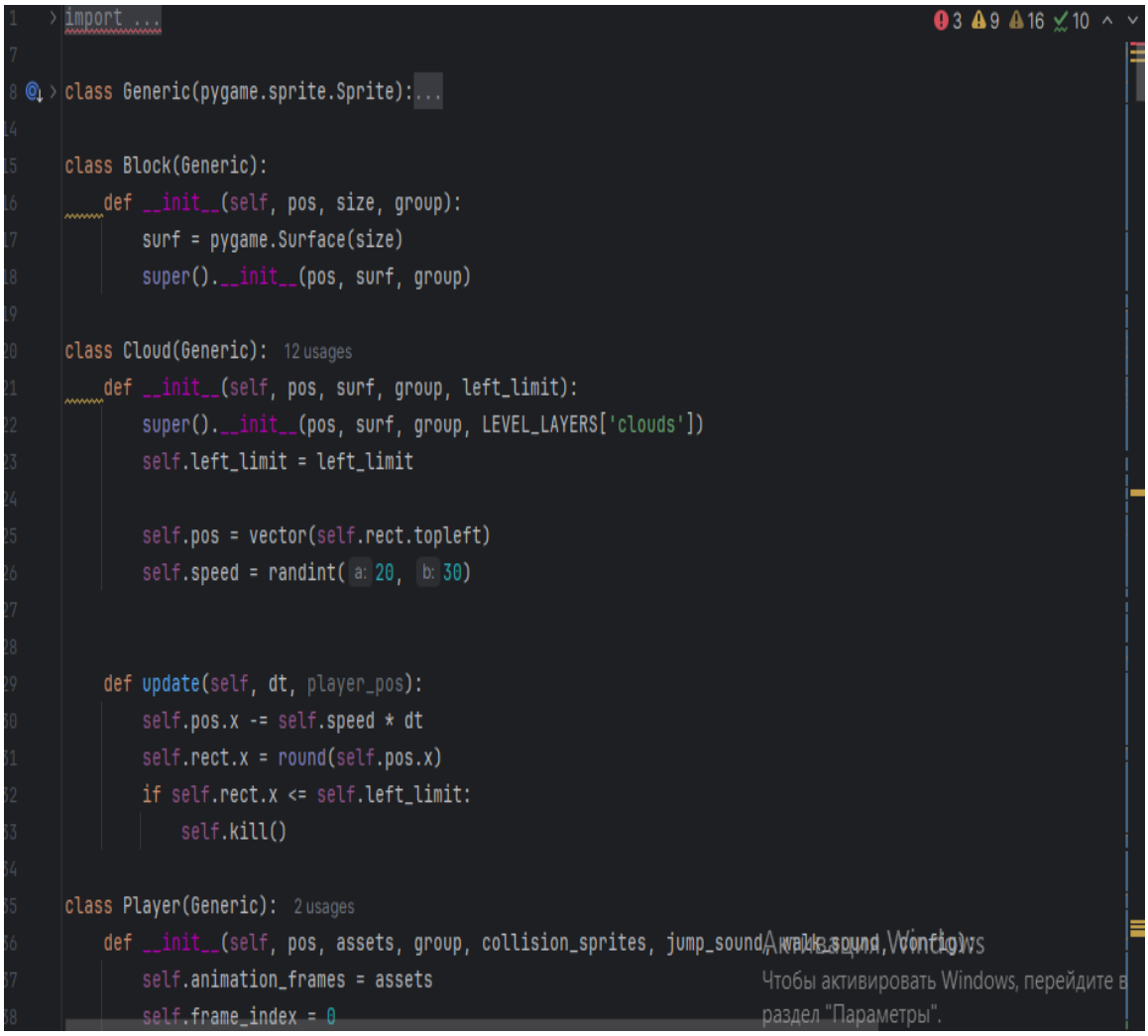
Додаток А

MAIN

```
class Transition:  1 usage
    def __init__(self, toggle):
        self.display_surface = pygame.display.get_surface()
        self.toggle = toggle
        self.active = False
        self.width = 720



    def display(self, dt):
        if self.active:

            self.width -= int(1200 * dt)

            if self.width <= 0:
                self.active = False
                self.width = 720
                self.toggle()
            pygame.draw.rect(self.display_surface, CURTAIN_COLOR, (0, 0, WINDOW_WIDTH, self.width), border_bott

if __name__ == '__main__':
    main = Main()
    main.run()
```

Додаток В

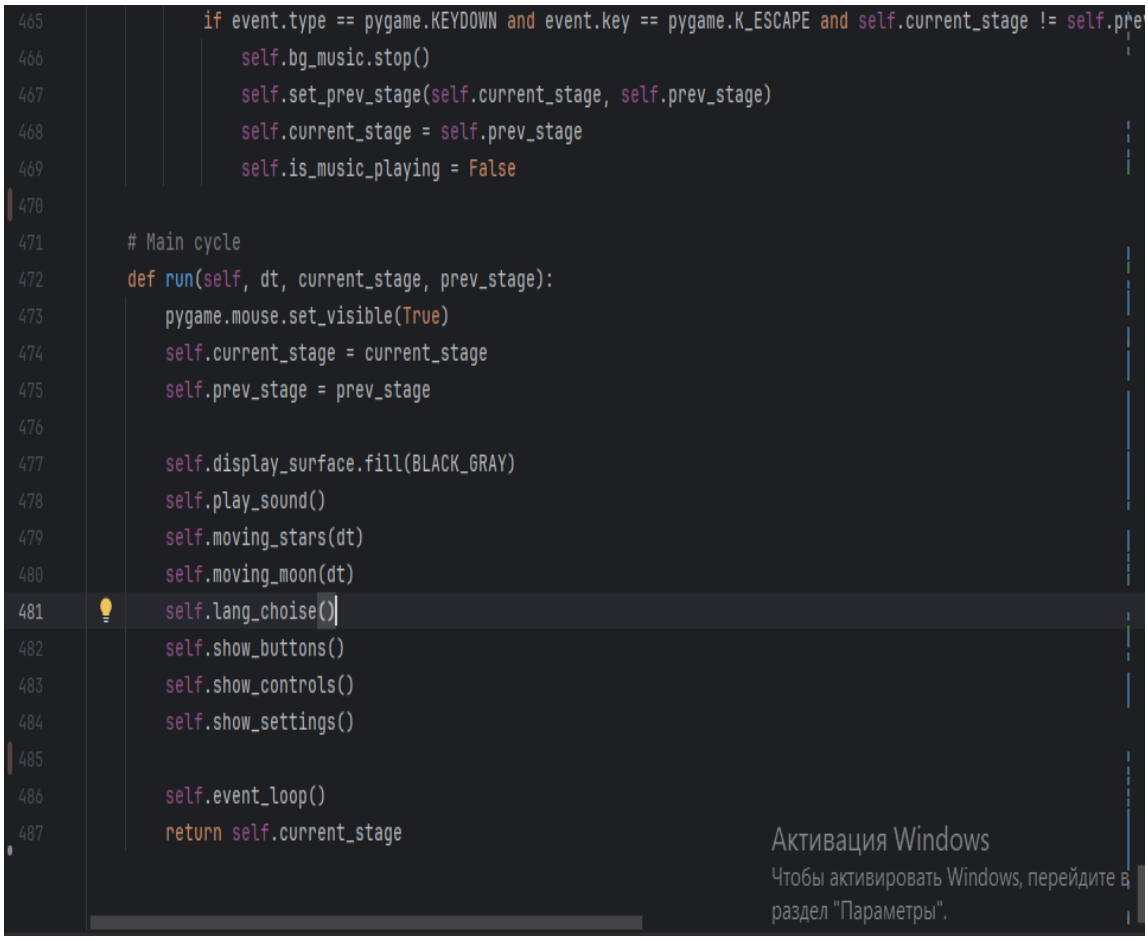
SPRITES



```
import ...

class Generic(pygame.sprite.Sprite):...

class Block(Generic):
    def __init__(self, pos, size, group):
        surf = pygame.Surface(size)
        super().__init__(pos, surf, group)

class Cloud(Generic):  12 usages
    def __init__(self, pos, surf, group, left_limit):
        super().__init__(pos, surf, group, LEVEL_LAYERS['clouds'])
        self.left_limit = left_limit

        self.pos = vector(self.rect.topleft)
        self.speed = randint( a: 20,  b: 30)


    def update(self, dt, player_pos):
        self.pos.x -= self.speed * dt
        self.rect.x = round(self.pos.x)
        if self.rect.x <= self.left_limit:
            self.kill()

class Player(Generic):  2 usages
    def __init__(self, pos, assets, group, collision_sprites, jump_sound, walk_sound, config):
        self.animation_frames = assets
        self.frame_index = 0
```

Додаток С

MAIN MANU



```
465             if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE and self.current_stage != self.prev
466                 self.bg_music.stop()
467                 self.set_prev_stage(self.current_stage, self.prev_stage)
468                 self.current_stage = self.prev_stage
469                 self.is_music_playing = False
470
471     # Main cycle
472     def run(self, dt, current_stage, prev_stage):
473         pygame.mouse.set_visible(True)
474         self.current_stage = current_stage
475         self.prev_stage = prev_stage
476
477         self.display_surface.fill(BLACK_GRAY)
478         self.play_sound()
479         self.moving_stars(dt)
480         self.moving_moon(dt)
481         self.lang_choise()
482         self.show_buttons()
483         self.show_controls()
484         self.show_settings()
485
486         self.event_loop()
487         return self.current_stage
```

Додаток D

MANU

```
class Menu:  2 usages
    """Class for upper menu in the editor's mode."""
    def __init__(self):
        self.display_surface = pygame.display.get_surface()
        self.create_data()
        self.create_buttons()

    def create_data(self):  1 usage
        self.menu_surfs = {}
        for key, value in EDITOR_DATA.items():
            if value['menu']:
                if not isinstance(value['menu_surf'], tuple):
                    if not value['menu'] in self.menu_surfs:
                        self.menu_surfs[value['menu']] = [
                            (key, load(value['menu_surf']))]
                    else:
                        self.menu_surfs[value['menu']].append(
                            (key, load(value['menu_surf'])))
                else:
                    terrain_choices = []
                    for image in value['menu_surf']:
                        terrain_choices.append(load(image))
                    if not value['menu'] in self.menu_surfs:
                        self.menu_surfs[value['menu']] = [
                            (key, terrain_choices)]
                    else:
                        self.menu_surfs[value['menu']].append(
```

Додаток Е

FIRST LANGUAGE

```
# Main menu buttons
CONTINUE_BUTTON = 'Продовжити'
NEW_GAME_BUTTON = 'Нова гра'
CONTROLS_BUTTON = 'Управління'
SETTINGS_BUTTON = 'Налаштування'
QUIT_BUTTON = 'Вихід'
MM_ESC = 'Esc - Головне меню'
MM_RIGHT = 'Стрілка вправо - рух праворуч'
MM_LEFT = 'Стрілка вліво - рух ліворуч'
MM_F5 = 'F5 - Швидке збереження'
MM_F9 = 'F9 - Швидке завантаження'
MM_TAB = 'Tab - Рюкзак'
MM_X = 'Ч чи Вверх - Використати/увійти'
MM_DOWN = 'Вниз - Присісти'
MM_SPACE = '  - Стрибок'
ED_MD = 'Перехід до режиму редактора'
GO_TO_ED = 'РЕЖИМ РЕДАКТОРА'
MM_VOLUME = 'Музика | Звуки'
MM_FULLSCREEN = 'Повноекранний режим'
MM_FLSC_ON = 'Ввімкнути'
MM_FLSC_OFF = 'Вимкнути'

INV_INSC = 'Спорядження'
KEY_FOUND = 'Знайдено'
KEY_NOT_FOUND = ''

GAME_OVER = 'Ти програв, страждай знову'
```

Додаток F

SECOND LANGUAGE

```
22
23  INV_INSC = 'Inventory'
24  KEY_FOUND = 'Found'
25  KEY_NOT_FOUND = 'Not found'
26  GAME_OVER = 'Game Over'
27
28  # End cutscene
29  THE_END = 'The End'
30  PROGRAMMER = 'Programming - LF'
31  GRAPHICS = 'Graphics - LF'
32  MUSIC = 'Music - Ivan LF'
33  SOUNDS = 'Sounds - mixkit.co/free-sound-effects/'
34
35
36
37  # HELPER SECTION
38  HLP = {
39      'has_item': 'Press "Up" or "X" to enter',
40      'yellow_not': 'Find the yellow key to unlock',
41      'pink_not': 'Find the pink key to unlock',
42      'green_not': 'Find the green key to unlock',
43      'hammer_not': 'Find the hammer to break the wall',
44      'hammer_yes': 'Press "Up" or "X" break the wall',
45      'sleep_well': 'Press "Up" or "X" to fall asleep',
46      'machine': 'Press "Up" or "X" to repair the machine',
47      'game_saved': 'Quicksaving',
48      'game_loaded': 'Quickloading',
49  }
```